Network Models of Pathogen Evolution: Reconciling Population Genetics and Epidemiological Dynamics

Kawashima, Kent Diel

Doctor of Philosophy

Department of Genetics
School of Life Science
SOKENDAI (The Graduate University for Advanced Studies)

Network Models of Pathogen Evolution: Reconciling Population Genetics and Epidemiological Dynamics

Kawashima, Kent Diel

Doctor of Philosophy

Department of Genetics

School of Life Science

SOKENDAI

(The Graduate University for Advanced Studies)

2018

ACKNOWLEDGMENT

I would like to express my utmost appreciation to my advisor, Professor Hiroshi Akashi, for being mentor that taught me how to think critically and work independently during my doctoral studies. I would like to thank you for not only encouraging me my research, but also for challenging me and allowing me to grow as a research scientist on my own terms. I am grateful for teaching me how to write better reports, give better presentations, and helping me think like a scientist. Thank you to Professor Akira Sasaki, Associate Professor Kazuho Ikeo, Professor Tetsuji Kakutani, Professor Akatsuki Kimura, and Professor Yasukazu Nakamura for serving on my doctoral thesis examination committee. Thank you for the time you spared and the patience you expended to read and understand my work. I would like to acknowledge Ikeo-sensei, for serving as the chairperson for my examination committee as well as my previous progress report committees. Thank you for sparing your time to always come to my poster sessions and for providing candid comments that helped me better focus my research. I would like to especially thank Professor Sasaki for providing his expertise to my examination committee as the external member, as well as for his patience and understanding in critiquing and thoroughly correcting my manuscript. It is difficult to imagine finishing this work without his insight and support. It would be remiss not to thank my previous committee chairperson, Professor Ituro Inoue, and Professor Jun Kitano for serving previously on my progress report committee and sharing your insight which helped me grow as a researcher. I would also like to thank previous and current lab members and staff of the Division of Evolutionary Genetics for your support throughout these years. Thank you very much to Neha Mishra and Haruka Yamashita for your support in good times and in bad, to Assistant Professors Naoki Osada and Tomotaka Matsumoto for your time and understanding, to our lab secretaries Sei-san and Takeshima-san for always lending a helping hand. I would like to thank the people at NIG, from the professors who have shared their knowledge; the students for their friendship; and the staff for their hard work. I would like to extend a special thank you to the people at the General Affairs and Education Team for their patience in tirelessly reminding me of the deadlines and helping me file an innumerable amount of paperwork. I would also like to mention my Japanese language teacher, Miyauchi-sensei not only for teaching me

Japanese, but also for her tireless support and always looking out for me. A special thanks to all my friends who have always been there to listen to me gripe and help me keep on keeping on. Finally, a special thanks to my mother for making me the person I am today. Thank you for your unwavering support and your bottomless love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF SYMBOLS

A	number of triangles
c	clustering coefficient of a node
D	network density
d	number of degrees/connections per node
d_r	number of random edges to add for each new node during graph generation
$\bar{d}$	average number of degrees/connections per node
$\tilde{d}$	total number of undirected edges/connections in the network
$\tilde{d}_d$	total number of directed edges/connections in the network
e	natural number
F	pathogen fitness value
F'	normalized pathogen fitness value
F_{ST}	fixation index
f	site fitness value
E	number of exposed host individuals in the population
I	number of infected host individuals in the population
k	number of pathogens transmitted (number of migrants)
m	migration rate
N	total number of host individuals in the population (population size)
N_d	number of pathogens/individuals within a deme (subpopulation size)
N_e	effective population size
n	number of subpopulations (demes)
P_T	transmission probability given a connection is present
P_c	probability of creating an edge between two nodes during graph generation
P_A	probability of adding a triangle after adding a random edge during graph generation
p	allele/genotype frequency, *also probability*
R	number of removed host individuals
S	number of susceptible host individuals in the population
s	selection coefficient
t	number of pathogen generations
v	birth rate
w	death rate
β	infection rate
β'	effective contact rate
γ	removal rate
Δt_I	duration of infection
δ	proportion among all demes
λ_d	expected number of pathogens transmitted given a transmission occurs
ρ	relapse rate
σ	exposure rate

SUMMARY

Unlike free-living organisms, pathogenic viruses, bacteria, and fungi need to infect a host organism in order to reproduce and must continually transmit and colonize uninfected hosts to survive in the long run. This mode of survival creates reticulated evolutionary histories dependent on the network of hosts and incurs repeated population bottlenecks every time pathogens transmit from an infected host. In order to capture the complexity of pathogen evolution, processes occurring within the host and events happening between hosts in the population have to be considered. However, our current understanding of pathogen evolution is limited to observing evolution within single infections, or characterizing evolution over a host population. While these two views agree that pathogen evolution is largely driven by purifying selection, within-host studies have found little evidence to support the idea that positive selection of pathogens naturally occurs within hosts. Results from within-host studies appear to be at odds with finding of positive selection using sequences sampled from different individuals. If pathogens are rarely selected for their fitness advantage, how can we explain the signature of positive selection detected from host population samples?

In this study, I attempted to reconcile these conflicting observations using network models of pathogen evolution that integrate within-host processes of replication, mutation, and selection, with the between-host processes of transmission and host-host interactions. In combining processes that occur at different levels of biological organization, I was able to examine the effects of transmission and host network structure on the genetic evolution of pathogens. Due to the periodic infection and transmission pathogen lineages undergo, I found that these events dampen the effect of natural selection even when pathogens possess large fitness differences. Two main factors impeded the effect of positive selection in pathogen evolution. First, the time it takes for selection to significantly raise the frequency of a new mutation is too long compared to the duration of acute infections. This means that new mutations remain at low frequency throughout the infection unless the fitness advantage is extremely high, or the duration of the infection is significantly long. The population bottleneck that occurs every transmission is the second reason positive selection cannot operate efficiently.

Experimental studies have shown that transmissions tend to impose a harsh bottleneck on pathogens. Since pathogens transmit periodically, this means that bottlenecks occur frequently. My results indicate pathogen evolution is not sensitive to fitness differences as the fixation probability of mutations tends to be flat even at moderately high levels of selective advantage. This indicates that the periodic expansion and reduction in population size in tandem with the short intervals between bottlenecks inhibit the role of positive selection in pathogens. This suggests that transmission parameters could play a bigger role in the evolution of pathogen than natural selection.

Placed in the context of a host network, both the density and the structure of the network influences the observed evolution of pathogens across the host population. Results from simulations on networks showed that diversity is significantly reduced when the host network is sparsely-connected than when it is densely-connected. However, any type of network used to condition the potential paths of transmission always showed lower levels of diversity compared to an unstructured population. While the network model saw reduced diversity, fixation probability is also diminished because the network structure promotes differentiation between pathogen lineages. Networks therefore make it hard for an advantageous genotype to sweep the entire host population unless it is related to host immunity. When the new mutation changes the pathogen’s immunological profile such that is create a new serotype, then these kinds of changes are expected spread rapidly as the pathogen is granted an effectively immunological naïve host population to infect.

CHAPTER 1
INTRODUCTION

This project was undertaken to bridge the gap between epidemiology, which studies the spread and control of disease, and population genetics, which examines change of allele frequencies and genotypes over time within and between populations, to develop a unified understanding of the evolution of pathogens.

The initial motivation for this study was to understand how virulence evolves across the span of epidemics. While molecular evolution of infectious disease pathogens has been observed in recurrent epidemics and pandemics including Influenza (Holland et al., 1982; Reid, Fanning, Hultin, & Taubenberger, 1999; D. J. Smith et al., 2004), tracking the genotypic evolution of pathogens over the course of outbreaks is a recent phenomenon facilitated by the development of new technologies like deep next generation sequencing and advancement in probabilistic single nucleotide variant calling (Jombart, Eggo, Dodd, & Balloux, 2011; Schreiber et al., 2009; G. J. D. Smith et al., 2009). Previous outbreaks appear to show changing infectivity and viral virulence during the spread of the disease, as if the pathogen was evolving as it was transmitting from person to person. This phenomenon has been observed in the SARS outbreak of 2003 (Anderson et al., 2004; Chinese SARS Molecular Epidemiology Consortium, 2004; Zhang, Wei, & He, 2006), the MERS outbreak of 2012 (Cotten, Watson, & Zumla, 2014; Lu & Liu, 2012), and the Ebola outbreak of 2014 (Ladner et al., 2015). Three factors influence the rate of infection for transmissible diseases: (1) the intrinsic virulence of the pathogen, (2) the susceptibility of the host, and (3) the host-host interactions that regulate transmission of the disease. These relationships, together with the epidemiological observations from previous epidemics, suggest that a feedback loop between processes occurring at two distinct scales of reference: processes affecting pathogen evolution within infected individuals and dynamics enhancing transmissibility and/or susceptibility to the infection as the disease spreads through the population. Indeed studies have shown that a feedback mechanism exists such that evolutionary processes can impact epidemiological dynamics (Antia, Regoes, Koella, & Bergstrom, 2003; Cen, Feng, & Zhao, 2014). Much less studied is the role of epidemiological factors such as the host network and transmission on the genetic evolution of pathogens.

I wanted to explore the relationship between epidemiological factors affecting the spread of the disease and the genetic evolution of pathogens that cause the disease. It is well-established that genetic changes can alter the pathogen phenotype and produce more or less virulent strains that affect the presentation of the disease (Frank, 1996; Lo, Tang, & To, 2006; Messenger, Molineux, & Bull, 1999; Nasser et al., 2014; Thrall & Burdon, 2003), widen the range of hosts the pathogen can infect (Bandín & Dopazo, 2011; Hall, Harrison, & Brockhurst, 2013; Lalić, Cuevas, & Elena, 2011; Woolhouse & Gowtage-Sequeria, 2005), and create new immunologically-distinct mutants that are can infect vaccinated individuals (Akira, Uematsu, & Takeuchi, 2006; Anderson & May, 1985; Pepin, Volkov, Banavar, Wilke, & Grenfell, 2010; Querec et al., 2009). Thus, genetic changes in pathogens invariably affect the size, range, and probability of spreading of diseases. However, much less is known about the effect of host-host and host-pathogen interactions on pathogen evolution. For instance, does host population structure affect genetic evolution of pathogens occurring within hosts?

Previous studies based on the metapopulation concept demonstrate a link between population structure and genetic evolution (Gladstien, 1977; Maruyama & Yasuda, 1970). Metapopulation models have shown that population subdivision can significantly affect the genetic diversity (Maruyama, 1970; Nagylaki, 1985; Pannell & Charlesworth, 2000; Wakeley, 2003) and fixation of existing and new mutations (Hartfield, 2012; Pollak, 1966; Slatkin, 1981; Vuilleumier, Yearsley, & Perrin, 2008). However, I found that existing theory and computational models were lacking. Existing population genetics models do not account for the cycle of infection and transmission unique to pathogens. While metapopulation models such as the finite-island model (Slatkin, 1977) can account for within- and between-host dynamics, some simplifying assumptions are incompatible with the dynamics of pathogen infection and transmission. Most classical metapopulation models (Levins, 1968, 1969; Maruyama, 1969; Slatkin, 1977; Wakeley, 1998) consider a constant metapopulation size which is inconsistent with the rapid increase and decrease of infections during an outbreak. Moreover, metapopulation models tend to assume random sampling of colonizers and does not always consider the spatial configuration of subpopulations. One way to incorporate more realism into existing the metapopulation framework is to use network theory. Network-based models have shown that the density of connections (Newman, 2002), the configuration of the network (Albert, Jeong, & Barabási, 2000; Meyers, Pourbohloul, Newman, Skowronski, & Brunham, 2005; Moore & Newman, 2000), and degree variance significantly affects the size and intensity of outbreaks (Eubank et al., 2004; Leventhal, Hill, Nowak, & Bonhoeffer, 2015; Salathe & Jones, 2010). However, these epidemiological models are blind to changes in the pathogen that could affect its phenotype and change the course of the epidemic.

OBJECTIVES

In this study, I wanted to connect population genetics with epidemiology in order to expand existing theory and to reveal the factors that affect the evolution of pathogens. To achieve these, I begin from a population genetics point of view by considering the genetic processes that underlie pathogen evolution within hosts during infections. Then, I develop a metapopulation model that incorporates a network structure of nodes and edges such that nodes are hosts and edges indicate routes of migration and colonization. To examine the dynamics of network metapopulation model, I created a new class of genetic simulator that is capable of simulating evolution at multiple scales in a flexible and efficient manner. Through numerical simulations, I determined how much within-host and between-host forces affect the trajectory of (new) mutations. I tested the effect of transmission size and infection length on the fixation probability of neutral, preferred and unpreferred genotypes. I also investigated whether the shape of the transmission network affects the spreading and overall fixation of mutations. I compared the fixation probability and site frequency spectrums when mutations occur in the regular, binomial and scale-free network configurations. Finally, I use regular, heterogenous, and superspreader transmission tree configurations to study the effect of irregular spreading on the fixation probability of mutations. Through these numerical experiments, I showed that both fitness differences and transmissions biased by network structure affect the frequencies of pathogens genotypes in the next generation.

BACKGROUND

Infectious diseases

An infectious disease is a communicable illness caused by the presence of a pathogen within the host as well as the host's response to the invading pathogen (Kawashima, Matsumoto, & Akashi, 2016). Upon entry into the host, the pathogen increases its numbers by redirecting resources to itself. For example, viruses invade cells and hijack the cell's enzymes to manufacture copies of itself. After a certain time, its presence and the ensuing damage within the host raises an internal host response to impede the spreading infection. It is during this stage of the infection that signs and symptoms of the disease become apparent. The time interval between exposure to the causative pathogen and the first symptoms of the disease is known as the incubation period. During this period, the infected individual may or may not be contagious depending on the type of disease and the individual's current state of health. Although difficult to assess, this lag time between when the individual notices the disease and the time the individual is contagious is an important aspect to consider in modeling the infection and transmission dynamics of the disease.

Infectious diseases can be classified into two categories according to their horizontal transmission pattern (Kawashima et al., 2016). Infectious diseases that spread without close contact of infected individuals are long-range infections while those that require direct or close-range contact are short-range infections. Cholera is a good example of a long-range infectious disease. Cholera is a water-borne disease that spreads through water contaminated with the pathogen *Vibrio cholerae*. On the other hand, short-range diseases have a limited range and may require close or direct physical contact with an infectious individual. In general, pathogens that transmit via contaminated airborne particles or expectorated droplets, and those that require contact to the skin or bodily fluids are short-range diseases. Distinguishing between these two categories is important because certain models and methods may be more applicable to one type and not the other.

Epidemics

Epidemics occur when the number of cases of the disease increases, often suddenly, in a community at a particular time (Centers for Disease Control and Prevention, 2012). Another term used interchangeably with epidemic is outbreak While outbreak carries the same definition as epidemic, it often describes an increase occurring in a smaller, more limited geographical area. Thus, an outbreak can become an epidemic if the disease spreads from the initially affected area into neighboring regions, broadening its reach.

Epidemics can be classified into one of four categories based on the manner of spread through the population: common-source, propagated, mixed, and other (Centers for Disease Control and Prevention, 2012). A common-source outbreak describes a scenario in which a group of individuals are all exposed from the same source and the disease is not passed on. A propagated outbreak on the other hand results from transmission of the disease from one individual to another. If the outbreak or epidemic is a combination of these two, it is called a mixed outbreak. In this case, it begins similar to a common-source outbreak but then spreads after to other individuals that were not exposed to the initial source. Finally, some outbreaks and epidemics may not necessarily fit under the previous three categories. For example, outbreaks of vector-borne diseases may result from a sudden increase of the vector population or increase in the prevalence of infection in the vector population. Note that epidemics and outbreaks are not limited to infectious diseases (Gregg, 2002). Exposure to toxins and radioactive substances can also begin outbreaks. Even the rise in the incidence of non-communicable diseases such as diabetes can be considered as an outbreak. However, in this study, outbreaks and epidemics refer to the spread of infectious diseases.

Epidemiological modeling

Epidemiological theory studying the infection and transmission of pathogens are founded on the compartmental model of Kermack and McKendrick (1927), the Reed-Frost stochastic chain binomial models (Abbey, 1952), and the Galton-Watson branching process (Galton & Watson, 1875).

These mathematical models attempt to describe the incidence and distribution of infections over time in homogeneous systems.

The compartmental model of Kermack and McKendrick (1927) is the basis for modeling dynamical epidemiological processes. Under this framework, a population infected by a microparasite, such as a viral, bacterial, or fungal pathogen, or protist, is divided into at least two groups – *susceptible* and *infected* – that track the number or density of hosts over continuous time (Hethcote, 2000; Kermack & McKendrick, 1927). Central in the original Kermack and McKendrick compartmental model is the transmission function βSI, where the transmission parameter $\beta > 0$. Note that the transmission function assumes that the rate at which susceptible individuals become infected is proportional to the numbers or densities of the susceptible S and infected I populations. This means that transmission is a "mass action process" in which the population is completely mixed, and individuals are not constrained by spatial structure similar to how chemical reactions are formulated. The original model classified individuals into three compartments – susceptible, infected, and removed – which specifies what is known as the *susceptible-infected-removed* or SIR model. Because the SIR model serves as the template for many other epidemiological models, it is important to be familiar this basic model.

SIR Model

When susceptible individuals become infected, infections are treated as instantaneous and infectious individuals enter the *removed* class R at a rate γ. Equivalently, this means that the time individuals stay infected is $1/\gamma$. The original formulation of the model has the form:

Equation 1. SIR model without vital dynamics

$$\frac{dS}{dt} = \frac{-\beta SI}{N} \tag{1.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I \tag{1.2}$$

$$\frac{dR}{dt} = \gamma I \tag{1.3}$$

where the total number or density of individuals in the population at a particular time t given by $N = S(t) + I(t) + R(t)$. This SIR model does not consider any host turnover as a result of birth or natural death, known as vital dynamics. As a result, this model assumes that the population is closed. To add vital dynamics to the model, two rate parameters v and w that describe the birth and death rate respectively can be added to the model as follow:

Equation 2. SIR model with vital dynamics (birth and death)

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + vN - wS \tag{2.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I - wI \tag{2.2}$$

$$\frac{dR}{dt} = \gamma I - wR \tag{2.3}$$

where $v = w$ such that the population size remains constant. When birth and death are taken into account, the model assumes that births produce individuals that are susceptible to the disease. On the other hand, the probability of dying from causes other than the disease is assumed to be random and to happen at equal rates regardless of the host's infection status.

Sometimes the removed class in the SIR model is also referred to as the *recovered* class, but this is not an accurate label given the dynamics of the system. The compartment following infection is called "removed" because individuals that enter are effectively detached from the dynamical system as they no longer have any effect on the system. Confusion arises because the removed class consists of individuals that may have recovered and now harbors complete immunity or individuals that have died. One way to solve this dilemma is to simply split the removed class into immune class Y, and dead class Z. The model now takes the form:

Equation 3. SIR model divided into immune and dead classes

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + \rho Y \tag{3.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - (\gamma_Y + \gamma_Z)I \tag{3.2}$$

$$\frac{dY}{dt} = \gamma_Y I \tag{3.3}$$

$$\frac{dZ}{dt} = \gamma_Z I \tag{3.4}$$

where the total number or density of individuals in the population at a particular time t is constant and given by $N = S(t) + I(t) + Y(t) + Z(t)$, but the number of living individuals is $S(t) + I(t) + Y(t)$ and varies over time. In this formulation, when complete and lifetime immunity is assumed, it is easy to see that both the immune and dead classes are dead-ends and do not participate in the dynamical process. Thus, the sum between the immune Y and dead Z classes in this divided model correspond to the removed class R in the standard SIR model such that $R(t) = Y(t) + Z(t)$ at all times.

One reason to explicitly differentiate between recovered and dead states is when the recovered cases do not experience lifelong immunity and can revert back to being susceptible. By splitting into separate recovered R and dead Z classes, recovered individuals will still contribute to the system by having waning immunity such that they lose at a rate ρ , after which they return to the susceptible class S (Equation 4). The rate of losing immunity ρ is known as the relapse rate. This model is known as *SIRS* and takes the following form:

Equation 4. SIRS model with explicit recovered and dead classes and loss of immunity

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + \rho R \tag{4.1}$$

$$\frac{dI}{dt} = \frac{-\beta SI}{N} - (\gamma_R + \gamma_Z)I \tag{4.2}$$

$$\frac{dR}{dt} = \gamma_R I - \rho R \tag{4.3}$$

$$\frac{dZ}{dt} = \gamma_z I\text{; normally this class is explicitly included} \tag{4.4}$$

where the total number or density of individuals alive in the population at a particular time t is given by $N = S(t) + I(t) + R(t)$. Normally, the last differential equation (4.3) is not explicitly modeled.

From these examples, it is straightforward to see the flexibility of the Kermack and McKendrick model and why it underpins the study of epidemiological dynamics. New compartments can be readily added to the model or existing compartments can removed, like in the SI model, or be split up to reproduce a particular phenomenon more realistically. However, because of mass action kinetics, this framework does not cope well with heterogeneity and cannot model stochastic processes (Hethcote, 2000). To see the full variety of compartmental model, Anderson and May's Infectious Disease of Humans (1992) contains an exhaustive listing of epidemiological models and their applications.

SIS Model

The susceptible-infected-susceptible or *SIS* model describes the spreading and maintenance of the disease in the host population where the duration of immunity is extremely short, or immunization does not happen. From the point of view of the SIR model, the SIS model is a simplification that removes the removed class R such that infected individuals become susceptible again by some recovery rate γ (Equation 5). However, unlike the SIR model, the SIS model considers the possibility of reinfection even after having the infection. This means that the SIS model does not undergo the same "boom-bust" dynamics that characterizes the SIR model. The model follows the form:

Equation 5. SIS model without vital dynamics

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + \gamma I \tag{5.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I \tag{5.2}$$

where the total number or density of individuals in the population at a particular time t given by $N = S(t) + I(t)$. This set of differential equations indicate that the frequency of the number of susceptible and infected cases could reach a stable non-zero equilibrium where the number of susceptible and infected cases do not change over time. When a disease reaches a non-zero equilibrium in the population, the disease is said to be *endemic*.

SEIS Model

The susceptible-exposed-infectious-susceptible or *SEIS* model is a more elaborate version of the SIS model. The main difference between the SIS and the SEIS model is that SEIS distinguishing between the state of being exposed to the disease-causing pathogen (Equation 6.2) and having the ability to transmit the disease (Equation 6.3), as opposed to instantaneously being infectious upon inoculation (Equation 5.2). Because of this additional compartment, the rate variable σ controls the lag time between inoculation and infectiousness by $1/\sigma$.

Equation 6. SEIS model without vital dynamics

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + \gamma I \tag{6.1}$$

$$\frac{dE}{dt} = \frac{\beta S}{N} - \sigma E \tag{6.2}$$

$$\frac{dI}{dt} = \sigma E - \gamma I \tag{6.3}$$

In some infectious diseases, the infection could first exist in a latent state such that the host appears to be uninfected due to a lack of observable symptoms (Kawashima et al., 2016). During this time, the latently-infected host may not be infectious but will become infectious at a later point in time. One the other hand, hosts in the infectious compartment are infected and can transmit the infection. Without distinguishing these two states, the rate of new infections will appear earlier and faster than the actual because of an overestimation of infected hosts that can transmit the disease. This differentiated treatment between exposed and infected cases is not limited to SIS-type infections and also occurs in SIR-type and other types of outbreaks.

Viral evolution theory: standard population genetics v. quasispecies theory

Viruses are thought to occupy a special niche in biology. Because they lack the requisite machinery to metabolize and reproduce independently, viruses are passive assemblies of biomolecules and are not considered living organisms that are often considered special entities. Because of this, viruses may not necessarily operate under the same laws governing biological processes in living organisms. However, in terms of population genetics, viruses can be treated just like any other asexually-reproducing living organism. Viruses also carry genetic material, either in the form of DNA or RNA, like living organisms. Viruses also have a set of genes that encode proteins to make a new copy of the virus. Therefore, viruses are still subjected to the same processes of mutation, selection and genetic drift. Although this much is true, viral populations appear to behave in ways that existing population genetics theory cannot fully explain (Domingo, 2002). Research suggest that viral evolution may be influenced by the amount of diversity present in the infection such that more diverse populations tend to be more evolvable (Burch & Chao, 2000; Codoñer, Darós, Solé, & Elena, 2006; Crotty, Cameron, & Andino, 2001). Concepts of group selection and the quasispecies theory have been invoked to explain these observations. However, it has also been shown that viral evolution can be explained under existing population genetics theory and some feel that the quasispecies theory is simply a reformulation of existing concepts (Holmes & Moya, 2002a; Wilke, 2005).

The quasispecies theory explains the evolution of an infinite population of theoretical organisms that replicate asexually and have extremely high mutation rates (Eigen & Schuster, 1977). Mathematically, the quasispecies theory focuses on the sequence space of $\mathbb{Z}$ possible genotypes where there are n_i organisms with genotype i. Assuming asexually reproduction produces A_i offspring for each genotype i, and the mutation rate from genotype i to j is q_{ij}, then the expected fraction of mutant offspring j is $w_{ij} = A_i q_{ij}$. Taking all genotype transitions into account, w_{ij} becomes matrix W whose eigenvector dictates the equilibrium distribution of genotypes. For example, consider for a sequence consisting of two biallelic sites whose value can be 0 or 1. In this configuration, the complete sequence space $\mathbb{Z}$ is composed of 4 possible genotypes – 00, 01, 10, and 11. Assume that genotype 00 never mutates and always produces one copy of itself while 01, 10, and 11 on average generates $1-x$ copies of themselves and x mutants for each of the other genotypes, where $0 \leq x \leq 1$. This scenario creates the quasispecies matrix W:

Equation 7. Quasispecies matrix

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-x & x & x \\ 0 & x & 1-x & x \\ 0 & x & x & 1-x \end{bmatrix} \tag{7.1}$$

which takes on the following diagonalized form:

$$W' = \begin{bmatrix} 1-2x & 0 & 0 & 0 \\ 0 & 1-2x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1+x \end{bmatrix} \tag{7.2}$$

Among the eigenvalues in 7.2, only the eigenvalue 1 and $1+\mathrm{x}$ are valid because they produce positive values. As long as the initial population was not solely composed of genotype 00, this means that as the number of generations t increases, the eigenvalue will become $(1+\mathrm{x})^t$. Since the corresponding eigenvector for this eigenvalue is $[0 \quad 1 \quad 1 \quad 1]$, the quasispecies model predicts that the population will eventually consist of genotypes 01, 10, and 11 and genotype 00 will be lost at equilibrium. The quasispecies aspect of the system emerges as a result of the limited transitions between genotypes such that the distribution of genotypes as a whole, rather than the single individuals, becomes the target of selection (Bull, Meyers, & Lachmann, 2005).

Quasispecies theory is often used to describe the evolution of viruses, particularly RNA viruses because their high mutation rates and large population sizes is thought to enable the population to cover a large portion of the sequence space, and endow it with special characteristics (Domingo, 2000; Lázaro, Escarmís, Domingo, & Manrubia, 2002; Stern et al., 2014; Vignuzzi, Stone, Arnold, Cameron, & Andino, 2006; Wilke, 2005; Wilke, Wang, Ofria, Lenski, & Adami, 2001). RNA viruses rely on a replicase called the RNA-dependent RNA polymerase that is known to have low replication fidelity (Sanjuán & Domingo-Calap, 2016). Replication fidelity describes the error rate of replication or the enzyme's intrinsic ability to copy genetic sequences accurately and determines the mutation rate. In fact, viral RNA-dependent RNA polymerase is very error-prone, with an error rate of up to 10^{-4} mutations per nucleotide for a single replication (Crotty et al., 2001; Sanjuán & Domingo-Calap, 2016; Sierra, Dávila, Lowenstein, & Domingo, 2000). This almost guarantees that at least one mutation is introduced for each new viral particle produced during replication.

The large census population sizes of viruses during infection is another aspect that make RNA viruses likely candidates for the quasispecies theory. During infection, the total number of viruses can reach up to 10^{12} particles in the whole body (Domingo, Sheldon, & Perales, 2012). Proponents of the quasispecies theory claim that this extremely large census population size in tandem with high mutation rates allow viruses to cover a large portion of the sequence space (Domingo et al., 2012; Lauring & Andino, 2010; Vignuzzi et al., 2006). This is an important point that underlies the fundamental characteristic defining the quasispecies theory. By having the ability to cover the sequence space, the population behaves like a connected set of genetic variants at the center of which is a master sequence with the highest fitness. Thus, the quasispecies theory predicts that the population exists as a "cloud" of genetic variants and posits that low fitness variants act to maintain mutational robustness in changeable fitness landscapes (Burch & Chao, 2000; Eigen & Schuster, 1977; Wilke, 2005). Due to this mutational coupling among individuals in the population, entities are dependent on each other and the population evolves as a single unit. As a consequence, selection no longer acts towards the single fittest variant but instead is applied on the entire distribution of variants. Recently, this has been referred to as "survival of the flattest" (Codoñer et al., 2006; Wilke et al., 2001) as opposed to the usual "survival of the fittest" owing to the observation that quasispecies that cover a larger range of the sequence space exhibits greater mutational robustness.

Through the lens of population genetics, the behaviors cited as evidence for the quasispecies theory in virus populations can also be explained by existing population genetics theory by asserting that the population is in mutation-selection balance (Goyal et al., 2012; Holmes & Moya, 2002a;

Jenkins, Worobey, Woelk, & Holmes, 1998). Mutation–selection balance refers to the equilibrium state where the rate at which new deleterious alleles created by mutation matches the rate at which deleterious alleles are eliminated by selection (Crow & Kimura, 1970). Consider a biallelic single locus in a haploid population with alleles A and B. Allele A is the wildtype allele with frequency p in the population of viruses in the infection and allele B represents a deleterious mutant with frequency q and has a fitness difference s relative to A. Suppose that A to B deleterious mutations arise in the population at a rate μ and the reverse beneficial mutation B to A occurs at a negligible rate. At each generation, purifying selection eliminates deleterious mutants reducing q by an amount spq, while mutation introduces new deleterious mutants increasing q by an amount μp. Mutation-selection balance occurs when these rates cancel each other out and q remains constant from generation to generation such that $spq = \mu p$. Therefore $q^* = \mu/s$ is the equilibrium frequency between mutation and selection. Based on this interaction, Holmes and Moya (2002a) pointed out when mutation rates are high enough, mutation and selection acting in concert on individual viral genomes can sufficiently explain the large genetic variability within current population genetics theory.

In a finite asexual populations, Muller's ratchet predicts that mutation-free individuals become increasing rare as they get lost to genetic drift over time (Muller, 1964). Muller's ratchet becomes a problem because purifying selection alone cannot completely prevent the accumulation of deleterious mutations (Chao, 1990). As individuals carry more and more deleterious mutations, the load of these mutations increases in a ratchet-like manner, decreasing population fitness and eventually leading to extinction (Andersson & Hughes, 1996; Duarte, Clarke, Moya, Domingo, & Holland, 1992; Felsenstein, 1974). Goyal et al. (2012) showed a way for viruses and other asexual populations to mitigate Muller's ratchet. Under their model, asexual populations require an influx of beneficial mutations to constantly compensate against mutation load. As Muller's ratchet intensifies, the model posits that the probability that a random mutation is beneficial increase. This assumes that compensatory mutations and back mutations that undo the original deleterious substitution become more common as the population fitness declines (Charlesworth, 2012). On the other hand, the model also predicts that the probability that a random mutation is beneficial decreases with increasing overall fitness. Thus a state exists such that a certain fraction of beneficial mutations maintains the absolute fitness of the population, acting as an evolutionary attractor (Goyal et al., 2012). This evolutionary attractor model shows that it is possible to explain the robustness of viral populations against Muller's ratchet without invoking quasispecies theory.

Another argument that skeptics raise is that the extremely large population size of viruses may actually not be sufficiently cover a large enough region of the sequence space to act as a quasispecies (Holmes, 2010; Holmes & Moya, 2002a). They argue that if neutral sites exist in viral genomes, this exponentially increases the neutral sequence space by factor equal to the number of neutral sites present. For instance, if 10 neutral sites exist in the genome, then the population has to be larger than 10^6 because the area of maximum fitness alone can have 10^6 master sequence variants due to random genetic drift. Thus, the population must be able to explore the surrounding lower fitness landscape for mutational coupling to occur and natural selection to act on the entire distribution of genetic variants. If neutral sites exist in viral genomes, even only 20 neutral sites would drive the minimum population size to greater than 10^{12}. If proven true, this directly questions one of the central assumptions of the quasispecies model.

Unfortunately, to show that RNA viruses form a quasispecies requires either observing a phenomenon that cannot be explained by population genetics alone or demonstrate that the observed features indeed occur as a result of the viral population operating as a quasispecies. Several pieces of evidence have been presented in support of viral quasispecies by showing mutational robustness as a result of the shape of the distribution of mutants (Lauring, Acevedo, Cooper, & Andino, 2012; Sanjuán, Cuevas, Furió, Holmes, & Moya, 2007; Stern et al., 2014; Vignuzzi et al., 2006) and evolutionary memory (Ruiz-Jarabo, Arias, Baranowski, Escarmís, & Domingo, 2000; Ruiz-Jarabo, Miller, Gómez-Mariano, & Domingo, 2003). However, this topic has remained contentious even as the term "quasispecies" has become more widely and loosely used, especially on populations with extensive genetic variation and high mutation rate (Domingo, 2002; Holmes, 2010; Holmes & Moya, 2002b).

In this work, I assumed that existing population genetics theory is sufficient to explain virus evolution and used population genetics concepts as the basis for simulating virus evolution within hosts. To include factors that influence the migration and dispersal of viruses between hosts in the population, I based my work on metapopulation models which will be discussed in detail in the next section.

Infections as a metapopulation

The concept of a metapopulation was introduced to consider the reality that many species tend to form persistent subpopulations or demes in actual habitats (Levins, 1968, 1969). Initially, the metapopulation concept was used to describe the population turnover among different subpopulations: the birth of new subpopulations – called demes – through colonization and their death through extinction. While the meaning of a metapopulation has broadened to describe any type of configuration where discrete populations exchange genes through spreading, it is this original definition that is most closely related to pathogen evolution.

Unlike free-living organisms, pathogens need to constantly infect hosts in order to reproduce and ultimately survive. This unique condition creates isolated pathogen populations within infected hosts akin to the demic structure in a metapopulation (Thrall & Burdon, 1997). Considering each host as a deme, mutation and selection of pathogens during infection affect only the pathogen population within the host. Apart from within-host processes, causative pathogens can be spread to other susceptible hosts during infection via symptomatic behavior such as coughing or sneezing for example. The transmission of pathogens from an infected host to an uninfected susceptible host is similar to a migration process where a selected set of individuals emigrate from an existing population and colonize an empty deme. As an individual recovers from the infection, the number of infecting pathogens drastically decreases and is comparable to a catastrophic extinction as defined in the classical metapopulation model introduced by Levins (1968, 1969) and Slatkin (1977) and stochastic metapopulation model of Whitlock and Barton (1997).

Unlike a panmictic population, a metapopulation structures the population into discrete units which localizes the action of genetic processes and adds migration to the list of processes that can affect the patterns of neutral genetic variation. Thus, it would useful to study the effects brought about by population subdivision as well as processes of colonization, migration, and extinction. Slatkin (1977) indicated that these metapopulation processes produce two conflicting emergent properties. First, there is an excess of genetic drift in demes colonized by a small number of individuals which leads to increased differentiation among demes due to founder effects and population bottlenecks. As a result, the same processes that increase genetic drift and can also significantly decrease the effective population size of the metapopulation. In contrast, the movement of migrants and colonists across the metapopulation reduces the genetic differentiation among demes. Thus, there exists an equilibrium level of genetic diversity balanced by the introduction of new mutations and the rate of migration between demes in the metapopulation. Pannell and Charlesworth (2000) also highlights three aspects acutely affected by metapopulation processes. First, a metapopulation with extinction and recolonization of demes creates age structure among the subpopulations such that within-deme genetic diversity between demes is expected to be greater compared to a metapopulation without recurrent extinction. Second, the birth and death of demes is expected to reduce the levels of species-wide diversity according to several recurrent extinction models. Third, subdivided population in general and metapopulation with recurrent extinction in particular affects genetic differentiation among demes based on between-deme genetic variance estimated using F_{ST} (Wright, 1949). The effect of recurrent extinctions on species-wide genetic diversity can also be shown, albeit indirectly, by estimates of effective population size. However, as Pannell and Charlesworth (2000) point out, there are several different “effective population size” estimates and it is important to distinguish how each is affected by recurrent extinctions.

Population structure models can be classified into “discrete” and “continuum” models depending on clustering and demarcation of individuals (Pannell & Charlesworth, 2000). Examples of discrete population structures are “island” (Slatkin, 1977; Whitlock & Barton, 1997; Wright, 1931), “stepping-stone” (Maruyama, 1969), and “island-mainland” (Slatkin, 1977) models that assume discrete subpopulations. On the other hand, continuums models assert that individuals are continuously distributed in one- or two-dimensional space, which facilitates modeling of transitional populations (Barton, Etheridge, & Véber, 2013; Wright, 1943). In the case of pathogen evolution, the discrete models are more suitable depictions that represent evolution secluded within individual hosts.

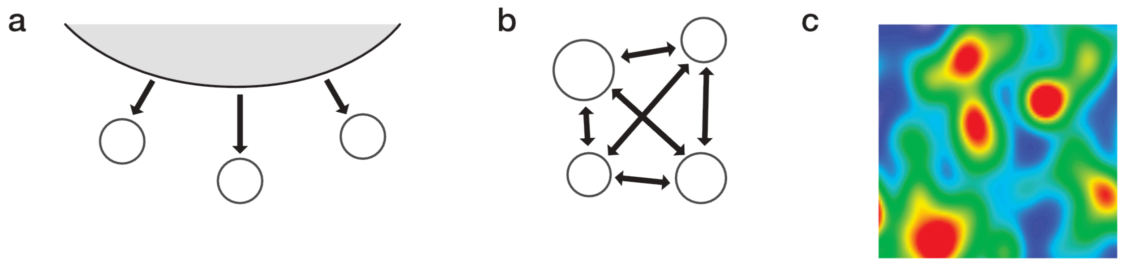


Figure 1. Discrete and continuous population structures.

Three types of discrete population structures (a-c). (a) Island-mainland model: circular areas indicate individual subpopulations with one large circular region depicting a continent. Movement is depicted by arrows between subpopulations. The "continent" or the "mainland" is a net exporter of migrants while islands are net importers. (b) Island model: each subdivision or patch is an island and experiences both incoming and outgoing migration. (c) Example of a continuous population structure model depicting population density over a geographical area, where red indicates high population density, blue indicates low density, and green shows areas of intermediate density. The key difference is that subpopulations are not isolated in demarcated regions and may overlap with one another.

In general, a discrete population structure assumes a finite number of demes linked by gene flow via migration (Figure 1b). Within each deme, individuals randomly mate and reproduce such that all individuals in the deme in the current generation are equally likely to be parents of the next generation. Thus, reproduction within demes follow the Wright-Fisher model where in a constant population size, each gene is expected to produce a single copy of itself in the following generation with a variance of one (Crow & Kimura, 1970). In the seminal work of Slatkin (1977) considering metapopulations with turnover, he presented two metapopulation models with extinction and recolonization. The first model assumed a single large deme as a net source of migrants (mainland or continental source) populating other demes that simply received migrants (islands). Under this model, the gene frequencies in the continental source are unaffected by the dynamics of the metapopulation of islands. The second metapopulation model he described is known as the finite-island model and is based on the island model proposed by Wright (1931) and studied by Maruyama (1970). Wright's island model defines a metapopulation of n demes, each containing a constant number of N_d diploid monoecious individuals. At each generation, a proportion m of genes are randomly selected to become immigrants and are uniformly distributed over the metapopulation. The island model definition serves as a template for a large number of classical metapopulation models of genetic variation (Pannell & Charlesworth, 2000). On the other hand, the finite-island model is simply an extension of the island model that describes a species composed of a set of n identical demes, each with a constant size N_d, that undergoes random extinction and immediate recolonization with new individuals at a rate e per generation (Slatkin, 1977). However, these simplified dynamics are often criticized as unrealistic and that the rate of extinction should depend on the size of individual demes (Pannell & Charlesworth, 2000). If deme sizes are heterogenous, large demes are less likely to go extinct and will be a net source of colonists while smaller demes will be constantly under the threat of extinction and are buoyed only by the immigration of individuals (Gaggiotti & Smouse, 1996; Pannell & Charlesworth, 2000). Given this scenario of recurrent extinction and recolonization, demes would have geometrically-distributed age structure such that for n demes and d proportion of demes that ne demes are one generation old, $nd(1-d)$ demes are two generations old, $nd(1-d)^2$ are three generations old and so on. Because extinction biases the age distribution of demes towards younger ages, metapopulation processes that affect the genetic composition of younger subpopulations will have a greater effect relative to those that are older. A generalized form of the island model was defined by Whitlock and Barton (1997) using backward migration matrix to indicate the probability that a gene in deme i came from deme j in the previous generation. Briefly, given a backwards migration matrix M; m_{ii} is the proportion of resident genes – genes in deme i that descended from genes also in deme i in the previous generation. In contrast, $\sum_{j \neq i} m_{ij}$ is the proportion of genes that are immigrants from other demes. The process of extinction and recolonization can be modeled using the using backward migration matrix. If deme i goes to extinction and is immediately recolonized, this can be represented by setting $m_{ii} = 0$ such that all individuals in the deme are populated by immigrants. Using this formulation, Barton and Whitlock (1997) (also Whitlock & Barton, 1997) found subdivision decreases the variance in reproductive success between individuals across the whole population which reduces the effect of drift.

There is abundant evidence demonstrating the effect of population structure on genetic differentiation based on estimates of genetic variance among subpopulations (F_{ST}) and indirectly, using effective population size. The fixation index (Wright, 1949), F_{ST}, estimates the extent of genetic differentiation between demes based on the correlation of two randomly-sampled genes from the same population compared to the correlation of when randomly selected from the entire population. Consider two alleles A and a are segregating at a single locus that have whole-population average frequencies $\bar{p}$ and $1-\bar{p}$ respectively, and $var(p)$ represents the variance in frequency of A across demes in the subdivided population, then F_{ST} is:

***Equation 8. Wright's F*$_{\mathrm{ST}}$**

$$F_{ST} = \frac{var(p)}{\bar{p}(1-\bar{p})} \tag{8}$$

A generalized version of F_{ST} for more than two alleles is Nei's G_{ST} (Nei, 1973). Both statistics range from 0 to 1.0 where zero means that the population is completely mixed and freely interbreeding and a value of 1.0 indicates completely differentiated populations. Statistically significant deviation from 0 indicates either divergence between demes caused by genetic drift, or as a function of selection due to differentially favored alleles in different demes. In the case of the island model without recurrent extinction, Wright (1949) showed that demes become significantly differentiated from one another when the product of the deme population size N and the migration rate m is $Nm < 1$. If $Nm > 1$, the chance that different alleles fix in different demes is low due to the effect of migration. Within subpopulations, genetic diversity is expected to be an unbiased estimate of overall metapopulation genetic diversity if all subpopulations are linked by migration, migrations do not change deme population size, and each deme contribute equally to the next generation (Nagylaki, 1982, 1998). Studies have also considered the effect of population subdivision on the effective population size in relation to an ideal panmictic population. Whitlock and Barton (1997) noted that there are many types of "effective population size" such as "variance effective size", "inbreeding effective size", "eigenvalue effective size", and "mutation effective size". Moreover, they showed that effective population size N_e may increase or decrease compared with the same unstructured population of the same size N such that $N = nN_d$, n is the number of demes and N_d is the population within each deme. In this scenario, there also showed population size N_e depends simply on the variance in fitness among subpopulations. That is, when subpopulations contribute equally to the next generation, the reproductive success of individuals simply follows a Poisson distribution. However, when demes contribute unequally to the next generation, they showed that subdivision may lead to a significant decrease in effective population size estimates. In general, subdivision causes variance in the reproductive success between individuals across the whole population to decrease.

Evolutionary epidemiology

It is interesting that the fields of population genetics and epidemiology have a lot in common yet have remained separate entities, only converging recently with the advent of new technologies facilitating the study of pathogen genetics and evolution (Pybus, Fraser, & Rambaut, 2013). For example, both population genetics and epidemiology examine processes of transmission and loss over time. In population genetics, we are concerned with the transmission of genetic information, in the form of alleles and their change in frequency over time. On the other hand, epidemiology studies the incidence and distribution of infectious diseases caused by pathogens and parasites. Moreover, population genetics and epidemiology both share a fundamental basis in mathematics and statistics, have established mathematical frameworks that describe their respective dynamical processes, and extensively use dynamical models to simulate real-world phenomenon. In population genetics, forward genetic simulations based on the Wright-Fisher or Moran model are used to study the process of evolution in fine detail. Whereas in epidemiology, Markovian compartmental models are used to model disease spreading and maintenance in a population. Thus, it is natural to wonder why these disciplines did not converge earlier.

One possible reason is the fact that the processes that population genetics are concerned with operate at a different scale to the processes that epidemiology study. Spatially, genetic evolution of infectious disease pathogens occurs during infection of individual hosts. In contrast, the incidence and distribution of diseases happens at the level of the host populations. Dynamical processes occurring at two levels of organization is not intuitive to imagine nor is it straightforward to construct a computational model. It is also possible to think that marrying the two fields did not make sense because the processes they study occur in vastly different timescales. Evolutionary timescales often amount to thousands, if not millions of years while epidemiological timescales can be as short as a few weeks. However, studies have shown that infectious disease pathogens, especially viruses, indeed evolve at the timescale of epidemics (Fields, Knipe, & Howley, 2013). In fact, the study of viral evolution during epidemics may have been the key driver that have brought these two fields closer together (Pybus et al., 2013).

The main objective of evolutionary epidemiology is to study the feedback between epidemiological and evolutionary processes. Epidemiological dynamics of disease spreading can affect how natural selection promotes or diminishes advantageous and deleterious genotypes in the pathogen population. At the same time, new mutations in the pathogen can result in better transmission or alter the mortality of the ensuing disease, which affects the nature of epidemiological dynamics. Several techniques have been developed to explore both the theoretical aspects and the real-world aspects of evolutionary epidemiology using simulations as well as data gathered from disease infections.

Evolutionary invasion analysis is a popular technique to study the feedback loop between epidemiological and evolutionary processes. Rooted in evolutionary game theory, this method attempts to find the best invasion fitness to survive. In evolutionary game theory, the goal is to find an evolutionary stable strategy that survives against all other variants. Invasion analysis works in a similar fashion. Given an invasion fitness, which quantifies the growth rate of a new mutation or variant (Metz, Nisbet, & Geritz, 1992), the goal is to find an allele that makes the invasion fitness of all other possible variants negative in a population dominated by the optimum allele (Maynard Smith

& Price, 1973). However, this approach assumes that epidemiological and evolutionary timescales are uncoupled such that epidemiological dynamics operates more rapidly relative to evolutionary change (Frank, 1996).

Phylodynamics is a more recent method that combines models of evolution and epidemiology to describe the phylogenetic history of pathogens in the context infectious disease dynamics. The term "phylodynamics" was coined by Grenfell et al. (2004) to describe the "melding of immunodynamics, epidemiology, and evolutionary biology" to understand "how pathogen genetic variation, modulated by host immunity, transmission bottlenecks, and epidemic dynamics" under a molecular phylogenetics framework. While applicable to any type of pathogen, RNA viruses best fit this method because these viruses rapidly accumulate mutations over a short period of time owing to their short generation time and high mutation rate. Since their generation time is shorter than the epidemiological time scales, transmissions affect the patterns of viral genetic variation to reveal aspects of epidemiological history (Volz, Kosakovsky Pond, Ward, Leigh Brown, & Frost, 2009). Viral genetic variation can also be affected by natural selection and can be used to study the evolution of virulence and shifting of antigenic phenotypes (Koelle, Cobey, Grenfell, & Pascual, 2006). Thus, phylodynamics is applicable to any "measurably-evolving pathogen" that accumulates one or more mutations every transmission.

Genetic simulation

Computational models are indispensable tools to examine dynamical processes in fine detail under tractable conditions. Models try to capture the complex behavior of real-world phenomenon through algorithms and equation that can be played and replayed in an infinite number of ways by running computer simulations. In population genetics and molecular evolution, computational models have been used to study the evolutionary processes of mutation, selection, recombination, and drift especially when analytical solutions are intractable. There are two approaches to simulate evolution in biological populations that differ in computational complexity and flexibility: a backward or coalescent-based method, and a forward strategy.

Backward or coalescent-based simulations are computationally efficient simulations that only consider the evolutionary history of surviving lineages. Coalescent-based methods are called backward simulations because they begin from the observed or extant population and then work backwards, ignoring individuals that do not belong to the lineage of extant individuals. While the basic coalescent model assumes the standard neutral model (Kingman, 1982), many types of coalescent models have been created since to simulate specific scenarios such as structured populations (Beerli & Felsenstein, 2001; Notohara, 1990) , variable population size (Griffiths & Tavaré, 1994; Tajima, 1989), evolution with recombination (Hey & Wakeley, 1997; Hudson & Kaplan, 1988), and selection (Kaplan, Darden, & Hudson, 1988; Neuhauser & Krone, 1997).

In contrast, forward simulations are less efficient as they simulate the entire population forward in time from past to present, hence the name. Although forward simulations require more computational power and is more complex than coalescent models, it offers greater modeling flexibility and more straightforward intuition. Unlike coalescent-based models that include only ancestors of observed individuals, forward simulations model all individuals regardless of whether they survive or go extinct. If examining the evolutionary process itself is the main objective, forward simulations should be preferred because it creates a complete picture of evolution of the population instead of only simulating the outcomes. The key advantage of forward simulations over coalescent models lies in the ability to model complex evolutionary scenarios. Forward simulations give the freedom to model selection and recombination, as well as include specific mating schemes and complex demographic scenarios (Carvajal-Rodríguez, 2008; Guillaume & Rougemont, 2006; Haller & Messer, 2017; Hernandez, 2008; Sanford, Baumgardner, Brewer, Gibson, & Remine, 2007). While some coalescent approaches can model specific types of recombination and selection, backward simulation cannot be easily adjusted to cope with slight deviations from the standard model. Forward models are also capable of simulating non-homogenous scenarios of selection and are not limited to specific parameter ranges (Wakeley, 2005). Therefore, forwards simulations tend to be easier to work with at the cost of higher computational time. This is clearly apparent from the abundance of forward genetic simulators built for general use, as well as for more niche applications.

CHAPTER OVERVIEWS

This dissertation is structured as a series of self-contained chapters each discussing its own set of topics. The first chapter serves as a general introduction outlining the overall theme and goals of this project, as well as a review of related literature that provides the necessary background to appreciate this work. Succeeding chapters each have their own introduction and background that features information specific that particular section. Likewise, each chapter after the introduction has its own methods section for experiments, and implementation details for new methodologies. Each chapter also includes its own set of results, discussion and conclusion sections. Finally, this dissertation ends with a conclusion chapter that summarizes the information discussed in the previous chapters and discusses the implications of this work. A short summary of each chapter follows.

Chapter 2. Contagion: A Stochastic Individual-based Model for Simulating Viral Evolution in Epidemics

In this chapter, I present a new method for simulating genetic information specifically designed for pathogens and other symbionts that rely on hosts for survival and create a program called Contagion that implements this algorithm. The novelty of this simulator lies in its concurrent simulation of evolutionary processes occurring during infection, and the spreading of pathogens from one host to another. To accomplish this, I developed a method that divides the genetic simulation into discrete encapsulations that can be computed in parallel to speed up computation time and reduce memory footprint. Prior to this work, existing programs can only model pathogen evolution within single infections, or the number of infections in a host population. With Contagion, evolution of pathogens spreading across a series of hosts in the population can now be simulated and examined. I validated the program by running unit tests to check whether individual components were working according to their expected behavior, as well as by running simulations and comparing the results to theoretical predictions.

Chapter 3. Host network topology affects the spread of new mutations

This chapter examines the evolutionary histories of pathogens and how it can violate assumptions in standard population genetics models. The key difference between the life histories of pathogens and free-living organisms lies in the reliance of pathogens for hosts to reproduce and survive. As a consequence, pathogens are captive to their hosts and can only be transmitted to other individuals that the host interacts with. I considered how this host-pathogen interaction regulates the probability of pathogens spreading and the probability that advantageous, neutral, or deleterious alleles fix. To understand how network effects changes fixation probability, I simulated different host network scenarios and their unstructured counterparts and compared the average fixation probabilities. I also examined whether the shape of the underlying host contact network affects fixation probability.

Chapter 4. Periodic infection and transmission parameters affects the fixation of mutations

In this chapter, I show that pathogen transmission is a key factor that constrains the evolution of pathogens. In cases of acute infection, I show that the length of time that pathogens are present within the host is insufficient for mutations to significantly increase in frequency before the next transmission event. Moreover, each transmission acts as a bottleneck to the existing pathogen population which decreases genetic diversity and eliminates rare variants. I show that recurrent bottlenecks with short intervening time intervals greatly diminishes the effect of intrahost selection.

Chapter 5. Virus Evolution in Alternating Hosts: Fixation Amidst Shifting Fitness Landscapes

This chapter examines the case of shifting fitness landscapes in the form of vector-borne disease pathogens. Vector-borne diseases such Dengue and Malaria are caused by pathogens that alternately live in two different host species. In the case of Dengue, the Dengue virus systematically infects the *Aedes aegyptii* mosquito vector and is transmitted to humans by a bite from an infected mosquito. However, the pathogen cannot be directly passed between humans and requires the mosquito vector to mediate the transmission.

Chapter 6. Conclusion

In this chapter, I summarize my findings and discuss its implications on our understanding of population genetics and molecular evolution of pathogens, specifically focusing on viruses. Avenues of future research are outlined.

CHAPTER 2
CONTAGION: A STOCHASTIC INDIVIDUAL-BASED MODEL FOR SIMULATING VIRAL EVOLUTION IN EPIDEMICS

INTRODUCTION

Unlike free-living organisms, pathogenic viruses, bacteria, and fungi need to infect a host organism in order to reproduce and must continually transmit and colonize uninfected hosts to survive in the long run. This mode of survival creates reticulated evolutionary histories dependent on the network of hosts and incurs repeated population bottlenecks every time pathogens transmit from an infected host (Turner & Elena, 2000). In order to capture the complexity of pathogen evolution, processes occurring within the host and events happening between hosts in the population have to be considered.

Forward population genetic simulations are often used to evaluate our understanding of complex evolutionary processes and test the power and limits of statistical methods. Forward simulations make it possible to program convoluted life cycles and create intricate demographic scenarios that capture the complexity found in natural populations, while being able to precisely examine evolution generation by generation. Although many simulation programs have been developed to recapitulate the evolutionary processes operating in nature (Carvajal-Rodríguez, 2008; Guillaume & Rougemont, 2006; Hernandez, 2008), most cannot model the host-dependent life cycle of pathogens and other symbionts.

Within a single host, pathogen evolution can be modeled similar to free-living organisms. We can assume that the population of pathogens within the host is a panmictic population. This is a simplifying assumption because studies have shown that even within the host, population structure exists due to cell, tissue, and organ specificities (Fields et al., 2013). To more closely replicate natural intrahost dynamics, more complicated metapopulation models (Murillo, Murillo, & Perelson, 2013; Thrall & Burdon, 1997; Vergu, Busson, & Ezanno, 2010) may be used to consider these subdivisions at the cost of increased complexity. Meanwhile, transmission events can be reflected in the model as migration events between demes. From the within-host population's perspective, new pathogens infecting the host is akin to inbound migration that contributes to the existing gene pool. Conversely, pathogens transmitted to other hosts become immigrants that will found new populations and create a new infections, or establish gene flow with other existing infections. Thus, if the objective is limited to examining pathogen evolution within a single infection, existing forward genetic simulators are capable of modeling these processes.

Understanding pathogen evolution beyond single infections involves considering not only the genetic evolution occurring during a single infection, but also examining the frequency and direction of transmissions across hosts in the population. Horizontal pathogen transmission can be imagined as a migration process that spreads radially and recurrently from an infected host that harbors a population of pathogens to other susceptible hosts. The route of transmission depends on the disease the pathogen produces and determines the set of susceptible hosts that can be directly infected. Unfortunately, existing genetic simulation software were not built to take these dynamics into account.

One way to track the transmission of disease is to use epidemiological simulators (Hethcote, 2000). This type of simulation software looks at a population of host individuals and simulates the spread of disease, tracking the infection status of each individual in the population over time. While epidemiological simulators offer a way to model the spread of the infection and track the transmission paths taken by pathogens, these simulators do not model genetic evolution and cannot be used to examine the frequencies of alleles or pathogen genotypes. Because of these limitations, studies that simulated pathogen evolution over several transmissions had to create their own custom-built forward genetic simulation software (Gordo, Gomes, Reis, & Campos, 2009; Murillo et al., 2013; Papaïx, Burdon, Lannou, & Thrall, 2014).

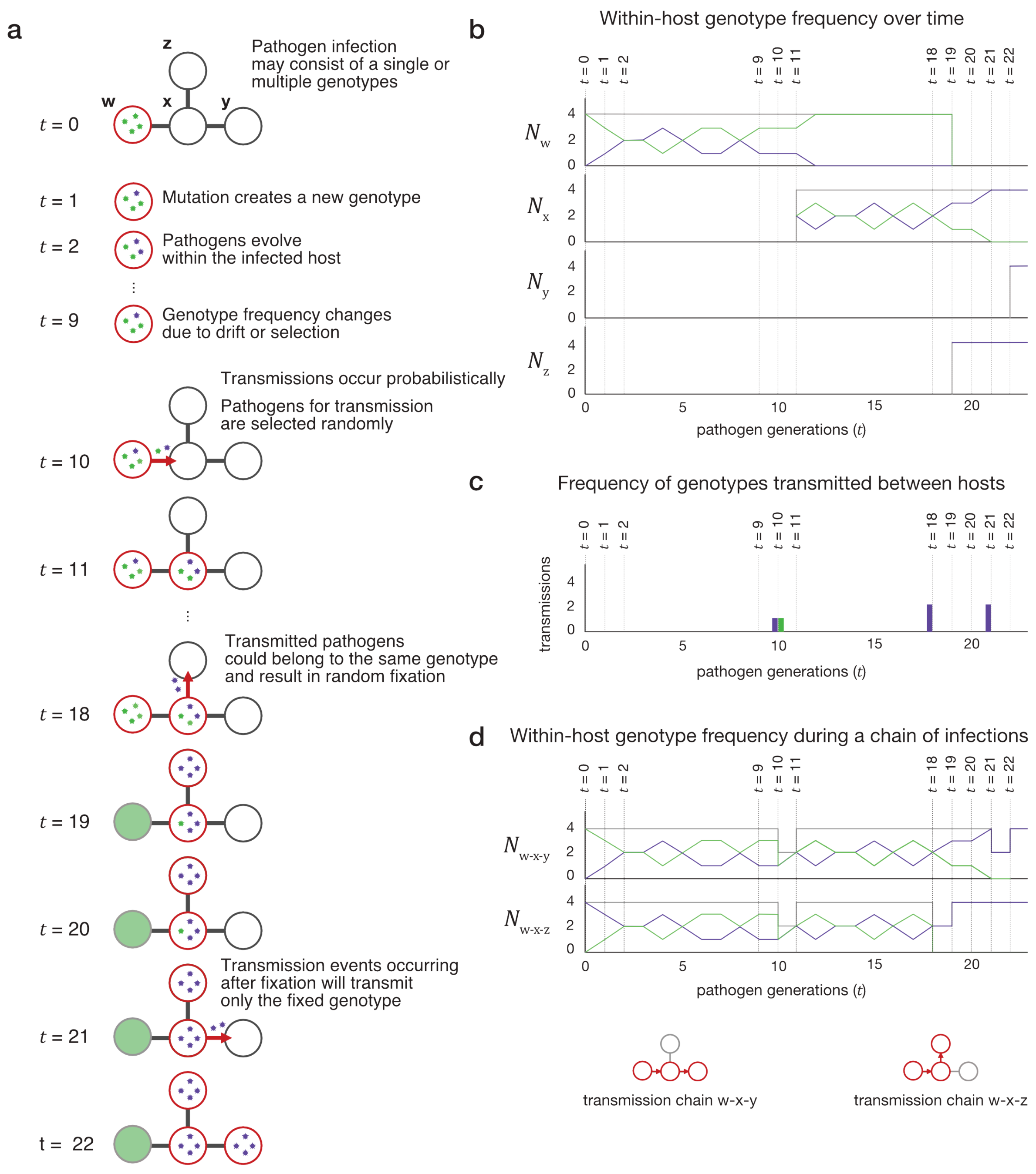
Here I present *Contagion*, a forward genetic simulator specifically designed to simulate pathogen evolution. Contagion is a hybrid simulator that explicitly models evolutionary processes at two scales of reference simultaneously. At the within-host level it acts like a regular forward genetic simulator that simulates the processes of replication, mutation, selection and drift on pathogen sequences. At the between-host level, it performs like an epidemiological simulator tracking the infection status of individual hosts and simulates pathogen transmission between hosts conditioned on the available routes present in the given host contact network. The novelty of this simulation method and software lies in the concurrent joint simulation of within- and between-host scales. By connecting these two scales, we can now examine how processes occurring at the within-host level can affect processes at the between-host level and vice versa.

In this chapter, I outline the underlying architecture behind Contagion and explain the computational tricks the software uses to handle massive amounts of simulated data. I also show examples of the types of analyses made possible by this new algorithm.

METHODS

Pathogen network evolution

The main idea behind this method is the concurrent treatment of processes occurring within the host and processes happening between hosts. The processes and interactions involved are illustrated in Figure 2. Within the infected host, a well-mixed population of pathogens is assumed, and genetic evolution occurs assuming discrete non-overlapping generations. At every pathogen generation, the processes of mutation, selection and genetic drift contribute to the evolutionary process. Between hosts, k pathogens from an infected host can be transmitted randomly to any susceptible neighbor provided an edge connects the two individuals. The total probability that transmission occurs between an infected host and susceptible host is defined by the product of the per-capita transmission rate P_T and the probability that the two hosts share an edge in the network. When a transmission event occurs, k pathogens from the within-host population (or λ_d expected number of pathogen when the number of migrants is stochastically modeled) are randomly selected with uniform probability to be transmitted. If these selected pathogens are inoculated into an uninfected host, the pathogens establish an infection and become founders of a new intrahost population, such that the size of the population increases from k to N_d pathogens in a single pathogen generation. On the other hand, it is also possible that the selected pathogens are transmitted to an individual with an existing infection. If the selected pathogens are inoculated into infected hosts, the migrants do not affect the pathogen population size of the recipient. This type of migration creates coinfection and gene flow, which facilitates mixing of pathogen genotypes that have arisen in different host individuals. To treat these processes concurrently, I assumed a discrete-time model where each unit time is a pathogen generation and within- and between-host processes moves forward by this much amount of time. This means that between-host stochastic processes are synchronized such that values indicate the probability or rate in terms of a single pathogen generation (Figure 2b). In actual time, one pathogen generation is estimated to be one 24-hour period .



a
Pathogen infection may consist of a single or multiple genotypes
z
w
x
y
t = 0
t = 1
Mutation creates a new genotype
t = 2
Pathogens evolve within the infected host
t = 9
Genotype frequency changes due to drift or selection
Transmissions occur probabilistically
Pathogens for transmission are selected randomly
t = 10
t = 11
Transmitted pathogens could belong to the same genotype and result in random fixation
t = 18
t = 19
t = 20
Transmission events occurring after fixation will transmit only the fixed genotype
t = 21
t = 22
b
Within-host genotype frequency over time
N_w
N_x
N_y
N_z
pathogen generations (t)
c
Frequency of genotypes transmitted between hosts
transmissions
pathogen generations (t)
d
Within-host genotype frequency during a chain of infections
N_w-x-y
N_w-x-z
pathogen generations (t)
transmission chain w-x-y
transmission chain w-x-z

Figure 2. Diagram depicting the dynamics of the pathogen network evolution model over time.

(a-d) The pathogen network evolution model extends the existing population genetics models to account for transmissions and other host-host interactions that affects the spreading of the disease. (a) In this model, pathogens evolve as they replicate within infected hosts. At the same time, infected hosts can transmit pathogens to directly connected neighbors. For example, transmission of pathogens from an infected host w to a susceptible uninfected host x at t=10 founds a new pathogen population within the individual in the next generation. Thus, each infected individual act like a deme in a metapopulation model. (b) As a result, infected hosts have individual pathogen evolutionary histories that can be monitored independently. (c) The evolution history of pathogens within infected hosts are initiated and strongly affected by the number of pathogens transmitted and the genetic make-up of the transmitting migrants. (d) Plotting the evolutionary history of pathogens by following

the pathogen population as it transmits between hosts provides a way to view genotype frequency changes within-hosts as the pathogens transmit from one host to another.

A key aspect of the pathogen network evolution model is the graphical framework used to describe the relationships between hosts in the host population. A graph is a collection of nodes or vertices and edges that connect them together. In the network pathogen evolution model, host individuals are represented as nodes in the network while interactions between individuals that facilitate transmission are represented as edges (Figure 2a). When an edge is present between two nodes, this indicates a direct transmission path between the two individuals. When one individual is infected with the pathogen and the other individual is susceptible to infection, the presence or absence of an edge conditions the transmission of the disease, regardless of other factors such as the number of migrating pathogens. This is an extension of the stochastic island model (Whitlock & Barton, 1997) that explicitly specifies whether migration between demes is possible or not, and if possible, at what rate.

IMPLEMENTATION

This section describes the technical details of the different parts of the program and explains how epidemiological and genetic concepts are implemented. Generally, the software works as a nested set of embedded data structures that may also contain still more embedded data structures. On the outermost level is an epidemic simulator that governs how transmissions occur. Within an epidemic simulator are (1) the set of data structures that model the host individual, (2) a network data structure that sets potential interactions among hosts, and (3) a tree data structure to keeps track of pathogen genotype frequencies and movement over the entire simulation time. And within each host data structure are pathogens data structures that represent its genotype. Simulations occur by running stochastic functions that modify the contents of these data structures according to a specified set of rules, also known as the simulation model.

Epidemic simulators

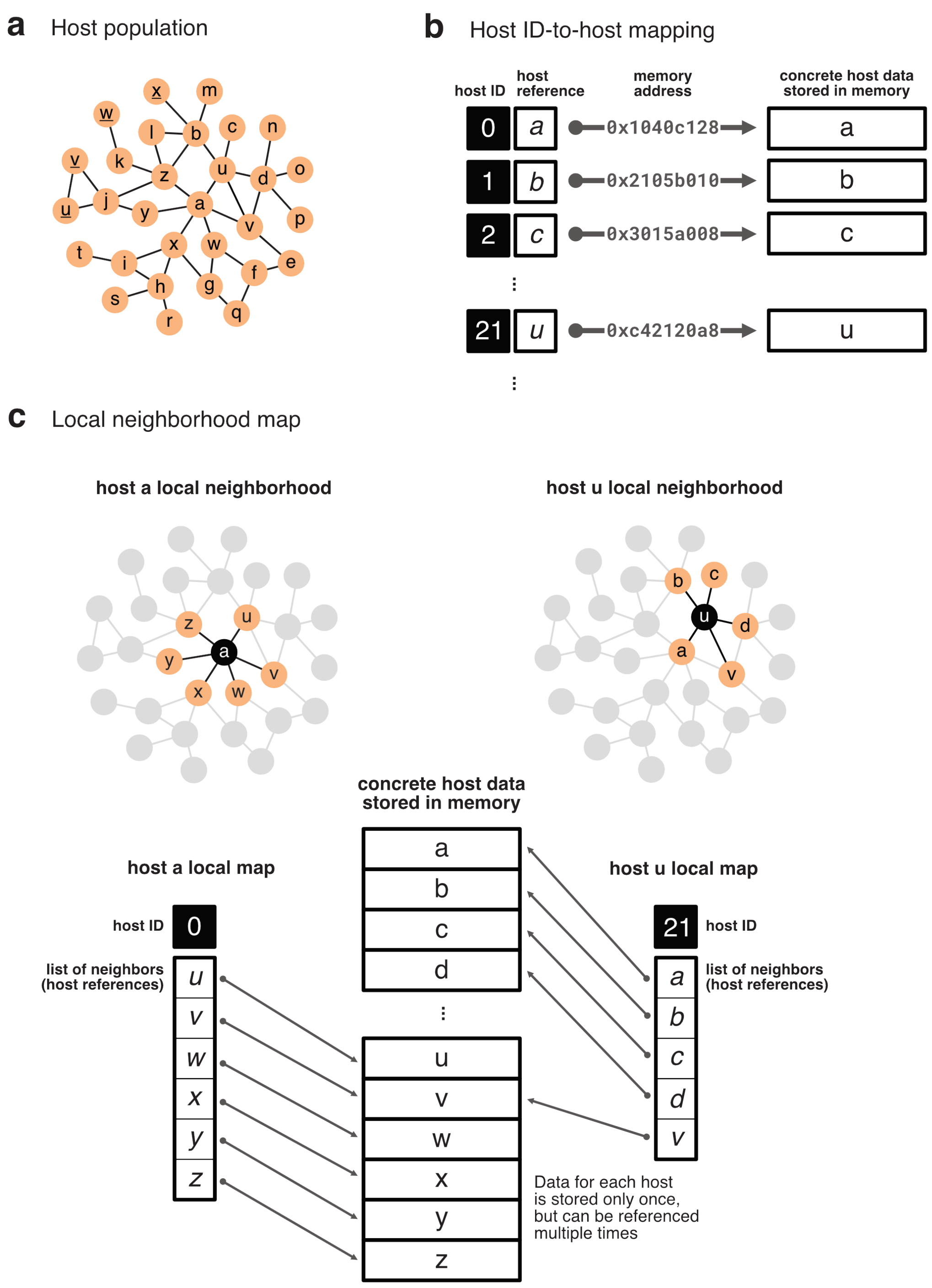
Different disease transmission and spreading behaviors can be simulated by using one of the many epidemic simulators implemented in the software. In Contagion, a simulator houses all the elements necessary to run a simulation and record its output. The simulator holds data structures that store (1) host references, (2) host status, (3) host local neighborhood information, (4) pathogen ancestry, and (5) models that list the rules for between- and within-host processes (Appendix II). Aside from holding data, a simulator also has associated functions, known as methods, that perform specific processes that can alter the state of the underlying data within the simulator, such as the infection status of hosts and pathogen genetic information. A complete list of simulator properties and methods can be found in the Appendix I, and the programming API in Appendix II.

Contagion was designed to be modular in order to facilitate a variety of ways to create new behaviors. There are two reasons for this construction: to make the program extensible to add new functionality, and to avoid redundancies in the codebase. For instance, the SI simulator is the most basic type of simulator and forms the basis for all the other more complicated epidemic simulators in the program (see Appendix II). New behaviors can be added or modified by simply writing new methods, or overwriting existing methods responsible for the current behavior, while keeping all the other extraneous parts intact. Moreover, the existing software can also be built on to create more elaborate implementations for anyone interested. This "do-not-repeat-yourself" or DRY technique lessens the amount of addition programming required to create new functionality and decreases duplicated code. More importantly, this helps projects with large codebases deal with errors and bugs in the program. For example, if a bug was found in a process shared across all simulators, then debugging is simplified because there will only be single point to repair as all the other simulators in the program inherit from a single implementation. An extensible design also helps other people to contribute to the project. Because Contagion is a free and open source software project, anybody can who can program in the Go programming language can implement new types of simulators by inheriting from the current list and add their own custom methods to get a particular functionality. The Contagion codebase can be downloaded from https://github.com/kentwait/contagion.

Simulating the spread of disease between hosts

In standard epidemiological models, the population of hosts are regarded as a well-mixed population where each host can transmit and be infected with the disease. Under this assumption, infection and transmission processes follow the law of mass action – the rate of new infections is directly proportional to the frequency or density of susceptible and infected hosts. When the population under consideration is large and lacks conditions that produce population structure, such as geographical barriers and social factors, then the law of mass action treats individual hosts like homogenously-mixing particles (as in chemistry) and provides a good approximation of the overall epidemic dynamics of the disease. However, in cases where hosts cannot be assumed to homogeneously interact with other hosts in the population, the law of mass action cannot be applied, and more sophisticated models are necessary.

Contagion can model outbreaks and epidemics under a wide array of population structures using a graphical representation to capture the interactions between hosts in a population. Here I explain the nested programming involved in detail (Figure 3). There are two data structures involved to represent the population of hosts: an unordered map indexed by a unique host ID to refer to each host individual (Figure 3b) and a network data structure that records the connection neighborhood for each host (Figure 3c). Consider a host population as illustrated in Figure 3a. In the simulation, the program refers to each host by a host ID as shown in Figure 3b. The host IDs for a population must begin at 0 and increment by one until all hosts have a unique ID. The is the first representation and is useful to randomly access any host in the program to retrieve information such as the host's internal state, or modify information given only the host's ID. For the network configuration of the host population, the program looks at a different representation. To illustrate this approach, take the group of hosts a, u, v, w, x, y, z highlighted in Figure 3c. From the viewpoint of a host, the only other hosts it needs to know are those in its immediate neighborhood because infections can only travel via direction connections. These directly-connected hosts form its local neighborhood. In the network in Figure 3, host a is directly connected to hosts u, v, w, x, y, and z and these form its local neighborhood. Local neighborhoods are stored in another unordered map whose key is the host ID of the reference point, and the value is a list of host references (Figure 3c). This enables quick access to any point in the network given a particular host as a focal point.



a
Host population
b
Host ID-to-host mapping
host ID
host reference
memory address
concrete host data stored in memory
0 a 0x1040c128 a
1 b 0x2105b010 b
2 c 0x3015a008 c
21 u 0xc42120a8 u
c
Local neighborhood map
host a local neighborhood
host u local neighborhood
concrete host data stored in memory
host a local map
host u local map
host ID
list of neighbors (host references)
Data for each host is stored only once, but can be referenced multiple times

Figure 3. Host population representation in the program.

(a) A host population can be imagined as a network where individuals are nodes connected to each other by edges. (b) To quickly examine hosts, a map of host IDs is used to refer to the concrete host data by reference (pointer reference). (c) The network structure of the population is encoded using a local mapping of directly-connected hosts for every host in the network. Because this map of local

neighborhoods refers to the host data by reference, two or more local maps can refer to the same host without each having an independent copy.

During the simulation, hosts are labeled according to their internal state of infection and can only have one type of label at any given point in time, similar to compartmental models in epidemiology (Figure 4). In compartmental models, there are a series of discrete compartments that are sequentially related to each other and function as categorical labels that classify hosts based on their infection history (Hethcote, 2000; Kermack & McKendrick, 1927). For example, in a SI model, there are two compartments – susceptible and infected. In this model, a susceptible host remains in the susceptible compartment until it is infected (Figure 4a). Once infected, the host transfers from the susceptible compartment to the infected compartment and remains infected until the end of the simulation. In the program, the type of epidemic simulator used to generate the simulation determines the number and label of compartments (see Appendix I for more information about epidemic simulators in Contagion). For example, an SI simulator depicts the SI model and has two compartments – susceptible and infected – whereas an SIR simulator depicts the SIR model and has three compartments – susceptible, infected, and removed. Currently, any of the compartmental models illustrated in Figure 4 can be used in program, but users can create more complex compartmental models by programming one from scratch or building on one of the existing models (see Appendix II for the Contagion API).

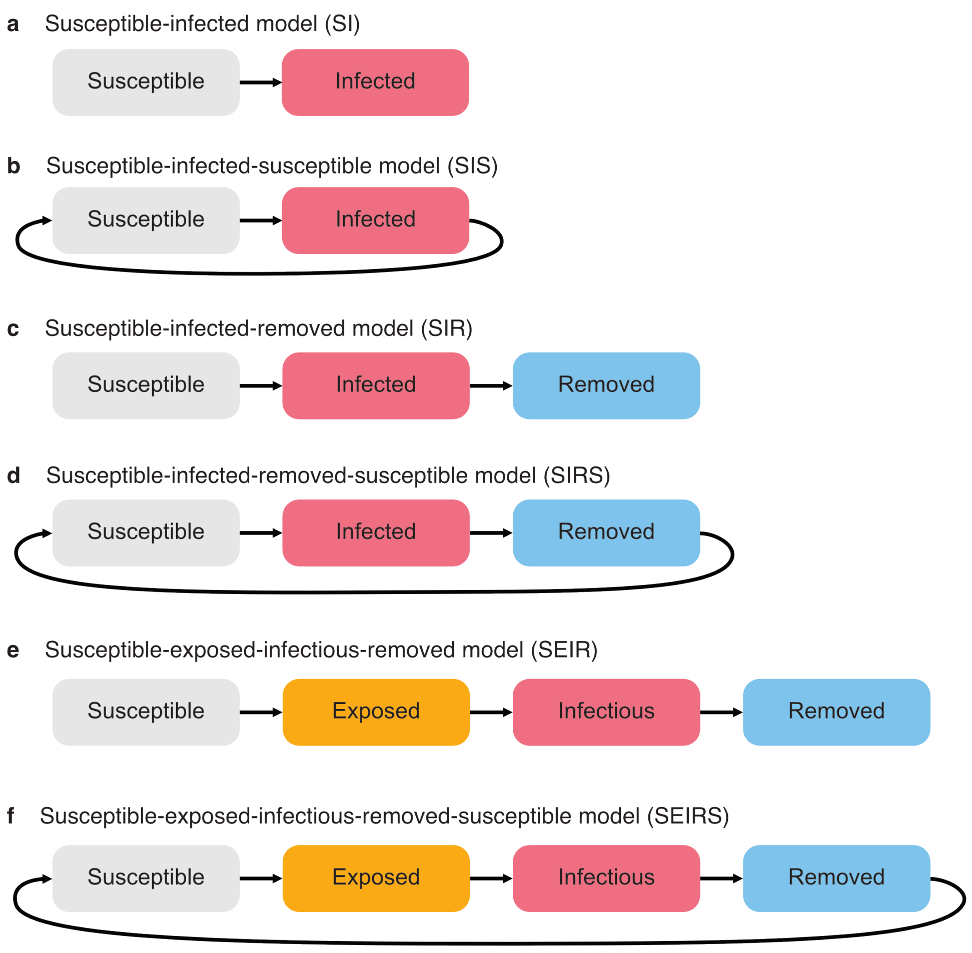

a Susceptible-infected model (SI)
Susceptible
Infected
b Susceptible-infected-susceptible model (SIS)
Susceptible
Infected
c Susceptible-infected-removed model (SIR)
Susceptible
Infected
Removed
d Susceptible-infected-removed-susceptible model (SIRS)
Susceptible
Infected
Removed
e Susceptible-exposed-infectious-removed model (SEIR)
Susceptible
Exposed
Infectious
Removed
f Susceptible-exposed-infectious-removed-susceptible model (SEIRS)
Susceptible
Exposed
Infectious
Removed

Figure 4. Common compartmental models used in epidemiology that are available in Contagion.

(a-f) Each box represents a discrete compartment used to classify the state of the host. Adjacent compartments lead into each other in a linear configuration, but more complex scenarios are possible. A host in one compartment moves into the next compartment given a particular rate over time such that it satisfies a Markov process. Exceptions occur when the waiting time is set to a constant instead of being sampled from an exponential distribution.

The current state of the host governs the processes the host can perform during each generation in the simulation. A susceptible host is uninfected and no processes are simulated for it except to wait for infecting pathogens. On the other hand, if a particular host is in the infected state, this host performs these four processes sequentially: (1) list down susceptible hosts in its local neighborhood, (2) for each susceptible host, perform a Bernoulli trial based for a given transmission probability to determine whether a transmission occurs, (3) if a transmission will occur, sample from a Poisson distribution using a given expected number of migrants to determine the number of pathogens to be transmitted, and (4) if the number of migrants is greater than zero, select migrants by uniformly sampling from the set of pathogens currently present within the infected host. In a full simulation, all these steps are performed. However, if only disease status of the host is being modeled, internal pathogens do not need to be considered and steps 3 and 4 are skipped and instead return a constant and a placeholder pathogen respectively. Depending on the epidemic simulator, an infected host may return into the susceptible state. In this case, an additional step is added to test whether the infected host will remain infected in the next generation. The number of simulation steps per host per pathogen generation depends on the number of compartments as well rules for each compartment.

Contagion precisely models network effects by encoding the local neighborhood of each host, the sum of which forms the network structure of the population. Moreover, this individual-based approach facilitates the simulation of other heterogeneities at the level of individual hosts, as well track the progress of the epidemic with knowledge of the exact transmission chain between patient zero and all subsequent infections.

This key feature of the program is also its biggest weakness. By modeling hosts individually, it takes a longer time and is more complicated to simulate epidemics compared to analytical methods that use arithmetic progressions. Moreover, the amount of data captured by the simulation can make populations in the order of tens of thousands with high network densities, the relationship between the number of nodes and edges present, prohibitively expensive to simulate in terms of memory and storage. However, population structure weakens at high network densities and makes network-based methods become superfluous because the analytical techniques that assume mass action kinetics can be used instead.

Simulating genetic evolution of pathogens within hosts

A key innovation of Contagion is the ability to concurrently simulate molecular evolution of pathogen genetic information over time. During the simulation, the genetic information of infecting pathogens can mutate and create new mutant strains that may have better, worse or equal replicative fitness compared to the parental genotype. To achieve this, the program also contains a forward genetic simulator that simulates the processes of replication, mutation, selection, and recombination that independently runs for every infected host in population.

State information

Pathogen genetic information is represented as a sequence of categorical states where each site can only have one state at any point in time (Figure 5). Moreover, the set of possible states must be consistent across all sites. This means that if the one site in the sequence has two possible states 0 and 1, the it is expected that all other sites in the sequence also only have the same two possible states 0 and 1. Apart from this constraint, categorical states are user-defined and can mean anything from individual nucleotides, codons, amino acids, segments of linkage disequilibrium, to any kind of genotypic state that can be represented as categorical variables. While the meaning of a site and the states it accommodates can be arbitrarily assigned, the number of categories (alleles) per site is technically limited to 256 states as each category is encoded as an unsigned 8-bit integer ranging from 0 to 255 in the simulation.

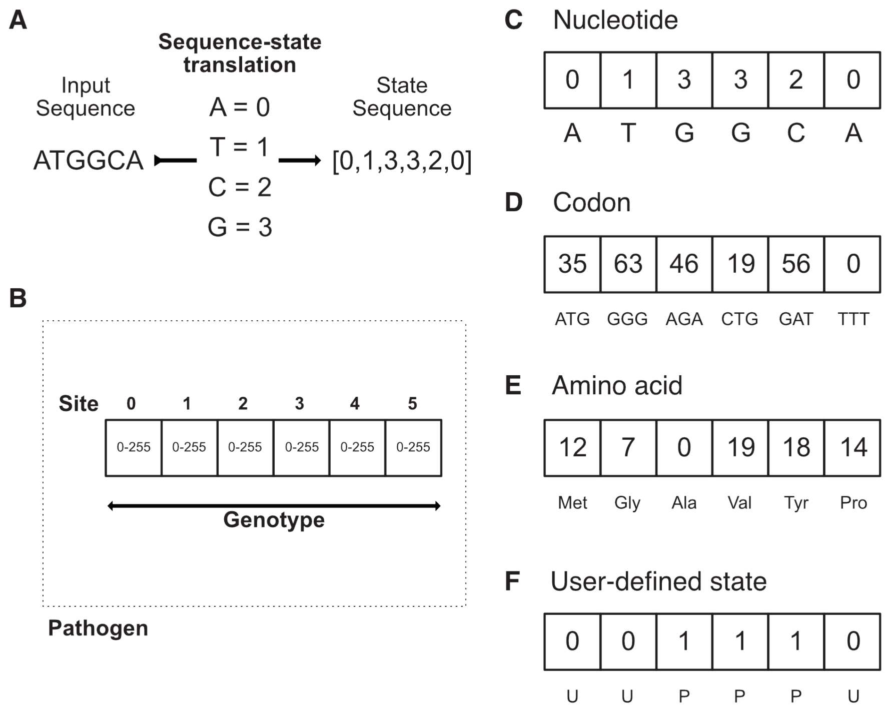
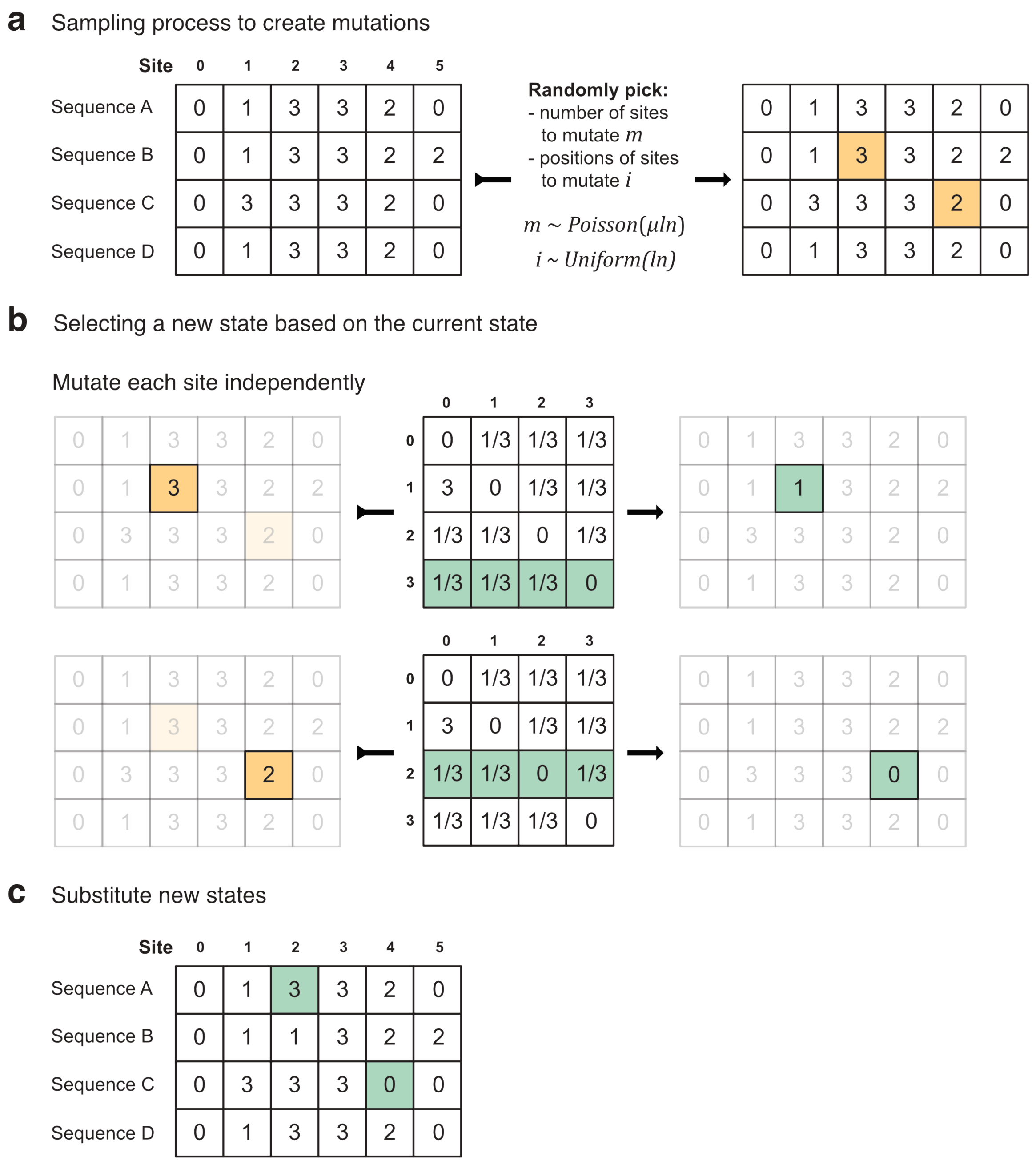


Figure 5. Depicting the pathogen, pathogen genotype, and site data models in Contagion.

(A) The program stores sequence information as a list of unsigned integers in memory and converts input into integers based on a given translation. (B) Each position in the list of states is called a "site", and the set of all sites in the list is the genotype and is encapsulated inside a "pathogen". (C) Each site in the genotype can take on any type of categorical value.

During the simulation, information may mutate due to random state transitions that occur in one or more sites in the sequence. This type of mutation is as a Markov process where the probability of transitioning from one state to another or to remain in the same state depends only on the current state of the site. The instantaneous transition rate matrix Q describes these probabilities for all possible state changes and stasis and can be used to determine the probability for one or more generations into the future. Thus, matrix Q is a square matrix whose side is equal to the number of categorical states a site can become. If each site can take on one of two possible alleles (binary states), then matrix Q must be a 2×2 transition matrix to describe all possible transitions. For nucleotide sequences, this means a 4×4 transition matrix is necessary because there are 4 possible bases per site. For amino acid sequences, a 20×20 transition matrix is required. For codons, a 64×64 transition matrix is required. Thus, if a site can take on one of n possible values, an $n \times n$ transition rate matrix is necessary in order to account for all possible transitions between all states.

As Contagion simulates each generation step in the simulation, only the immediately following state change needs to be considered. Given this approach, testing each site against matrix Q is computationally wasteful. To make this mutational process more efficient, the problem can be reformulated as a combination of two subproblems: (1) what is the expected number of sites that experience a state change in one generation, and (2) what is the transition probability to the new state given a state change occurs. Through this technique, the number of trials is reduced from each site in the sequence to a random subsample of sites that experience a mutation. To complete this reformulation, the transition rate matrix Q has to be refactored to create a new transition rate matrix $M = -dQ + I$ where M is the conditional transition rate matrix given that a substitution occurs, where $d = \text{diag}({q_i}^{-1})$ and I is the identity matrix. Instead of independently testing each site in the sequence against matrix Q to determine if the state changes or remains the same, Contagion first determines the number of sites that experiences mutation using a given mutation rate, randomly samples the sequence to select the sites to be mutated, and tests only those selected sites against the conditional matrix M to determine the new states. At low mutation rates, this method is significantly more efficient, but its advantage decreases as the mutation rate increases which leads more to sites experiencing mutation every generation.



a
Sampling process to create mutations
Site 0 1 2 3 4 5
Sequence A
Sequence B
Sequence C
Sequence D
Randomly pick:
- number of sites to mutate m
- positions of sites to mutate i
$m \sim Poisson(\mu l n)$
$i \sim Uniform(ln)$
b
Selecting a new state based on the current state
Mutate each site independently
c
Substitute new states

Figure 6. Simulating the mutational process using a conditioned transition rate matrix.

Diagram illustrates the mutation algorithm used by the program. (a) The number of sites over the entire pathogen subpopulation within the host is determined by sampling from a Poisson distribution whose mean is the product of the mutation rate, number of sites per pathogen sequence, and the *number of intrahost pathogens. Sites are then selected randomly for mutation. (b) The transition rate matrix determines the new identity of each site to be mutated based on the identity of the allele in the site and the transition probabilities to other states. (c) New states replace existing states in the pathogen sequences.*

Selection

Pathogens compete with each other using the concept of replicative fitness which defines the amount of new offspring a pathogen can create. In Contagion, the replicative fitness of a pathogen strain is based on the identity of its sequence and a fitness model. A fitness model is set of rules used to interpret sequence identities in order to assign a fitness value for a particular genotype. Currently, there are three fitness models implemented in Contagion: (1) multiplicative fitness matrix model, (2) additive fitness matrix model, (3) motif model.

The multiplicative fitness matrix model uses a predefined list of fitness values for each state at each site in the sequence and assumes that the contribution of each site to the pathogen's overall fitness is independent. In this model, the fitness F_m of the pathogen can be calculated by taking the product of all the fitness values f_i across all the sites from $i = 0, \dots, I$ such that $F_m = \prod_i^I f_i$.

Under the multiplicative fitness matrix model, the replicative fitness of a pathogen is relative to the fitness of other pathogens in the population, and the amount of new offspring a pathogen has is relative to a given population size. Therefore, the normalized replicative fitness of pathogens in the population can treated as the list of probabilities of sampling the pathogen, and the frequencies in the next generation with a population size N can generated by N independent trials each of which leads to picking exactly one of the pathogens in the current generation. This process describes sampling from a multinomial distribution. For example, the current pathogen population ($N = 10$) is composed of pathogens A, B, and C, and their normalized fitness are $\hat{F}_A = 0.5$, $\hat{F}_B = 0.4$, and $\hat{F}_C = 0.1$ respectively. To get the frequency of A, B, and C in the next generation with a population size of 10, the multinomial distribution is repeated sampled 10 times such that $X \sim Multinomial(10, \hat{F})$, where X is the vector of new frequencies for A, B, and C and $\hat{F}$ is the vector of normalized fitness values of the pathogen population. Given the relative nature of this process, this means that apart from calculating the fitness values of pathogens, a separate mechanism is required to determine the current population size.

The additive fitness matrix model is an alternative to the multiplicative model which uses absolute fitness values instead of relative rates. The additive fitness matrix model also uses a predefined list of fitness values for each state at each site in the sequence. However, unlike the multiplicative model, the contribution of each site to the pathogen's overall fitness is additive. In this model, the fitness F_a of the pathogen is equal to the sum of all the fitness values f_i across all the sites from $i = 0, \dots, I$ such that $F_a = \sum_i^I f_i$. Replicative fitness F_a is analogous to the absolute growth rate of the pathogen.

Selection occurs based on the number of offspring given by the sum of the individual site fitness values. Pathogens with larger absolute fitness produces more offspring than pathogens with lower absolute fitness. This means that if the mean absolute fitness of the pathogen population is large, then the population grows rapidly. As a side effect, the entire pathogen population can become extinct if all pathogens have an absolute fitness of zero. Due to the dynamics of this model, the population size does not have an intrinsic upper bound and can grow infinitely large. Two different behaviors can be set by the user to handle this possibility. When the pathogen population size within the host breaches a given threshold, the host can move from the infected state to the dead state mimicking how systemic infection can lead to death. On the other hand, the same threshold value can be used set an upper bound to the pathogen population size similar to the concept of ecological carrying capacity. If the population size of the next generation is greater than or equal to the threshold, the frequency of pathogens in the next generation can be computed through multinomial sampling by transforming the absolute fitness values into probabilities.

The third fitness model available in Contagion is the motif model. The motif model assigns fitness values based on the presence or absence of specific motifs in the sequence. The idea is to specify only sequence motifs that produce a selective advantage or disadvantage for the pathogen. A sequence motif is created by specifying a list of sites and target states, and the corresponding absolute fitness contribution if the motif is present. For each motif present, the model check for a match by comparing the states of specific sites between the motif and sequence. A match is identified only if the states in all sites are identical between the motif and the sequence. Thus, the probability of a match decreases as length of the motif increases. If the sequence did not match any specified motif, the fitness of the pathogen is set to a default value f_d which is the default grow rate of the pathogen without beneficial or deleterious mutations. If more than one motif was found, each of their effects on fitness is additive. Under this model, epistatic interactions between sites are accounted for, albeit in a limited manner.

Limitations

It is important to remember that the program considers a single site as the smallest unit of information in the simulation. This limitation is relevant when a single site in the simulation is made to represent codons or haplotype blocks which itself is composed of more than one mutable site. Take the case of codons for example. Each codon is three nucleotides long, with each nucleotide having the ability to independently mutate. Being three nucleotides long, recombination can occur at two sites within the codon as well as between boundaries of codons. However, if codons are used as the categorical variable in the simulation, mutations are treated as transitions between 3-character identities instead of individual nucleotides, and recombination becomes limited to sites between codons as individual sites are indivisible. Although it is possible to develop an implementation that considers intra-site changes, Contagion is heavily optimized based on the concept that sites are indivisible units of information. Given the amount of changes required to accommodate this feature, there is no plan to consider this option in the immediate future.

Currently, the sequence is assumed to have a constant length throughout the simulation to reduce computational complexity. This means the insertions or deletion mutations are not permitted in the simulation. The reason for this limitation lies in the way pathogen fitness is computed in some models. However, this may change in future release of the program.

Computational optimizations

Native multithreaded support

Contagion is built using the Go programming language to take advantage of concurrent computing features which enable several computations to be executed during overlapping time periods. This facilitates running computations in parallel and efficient scaling when multiple cores and multiple processors are available. To program with concurrency in mind, the simulation algorithms have to be composed such that calculations can be done independently and therefore can be performed in parallel. Mutating sites is one example of a type of computation in the simulation amenable to parallelism. Once sites for mutation have been identified, the substitution process is independent of each other. Therefore, it is possible to run the sampling process for each site in parallel, and then gather the results. Another process suitable for parallel computation is the selection process. The selection process takes a look at the normalized fitness value of each pathogen within the host and picks the next generation based on these values. Therefore, the selection process is concerned only with pathogens present within the host and is independent of selection occurring in other hosts, making it a good candidate for parallel computation.

The concurrent computing paradigm not only allows parallel processing, but also asynchrony – letting slower processes to run while the program does other kinds of work. This alleviates the problem of scheduling processes with different computational complexities and execution times. Concurrency in the Go language is derived from Hoare’s communicating sequential processes model which describes a concurrent system as component processes that operate independently and interact with each other solely through message-passing communication. In Go’s implementation of this model, concurrent processes are lightweight threads meant to perform short-running calculations and use channels to communicate and share data with each other. These lightweight threads are queued first-in-first-out and are distributed to open processing slots at runtime. This means even if the program creates 1000 threads for computing 1000 processes, if the computer only has 10 processing cores, then the first 10 threads are popped from the queue for processing, and only 10 processes are active at any given time. Thus, the larger the number of processing units, the more threads the computer can process simultaneously, and the faster the entire queue will be processed. If one thread finishes early, then the first thread in the queue takes the slot of the finished process and runs independently. Unlike in parallel computing, concurrency allows this kind of asynchronous event, making it an efficient way to process different types of work that may have varying execution times. In Contagion, the sampling processes in mutation, selection, and transmission are performed concurrently.

Efficient pathogen data modeling

Contagion uses an innovative mechanism to store pathogens in memory that allows the program to keep track of pathogen ancestry and reduce the memory footprint of the program compared to a brute force implementation.

In a straightforward simulation of pathogens evolving within infected hosts, each host infection can be considered as an independent forward genetic simulation that runs while the host remains infected. When overall genetic diversity is low, there will be multiple copies of pathogens with the same genotype within the same host and may also be found across different infected hosts. If the program naively stores each pathogen sequences in memory without dereplication, multiple copies of the same sequence will be stored and takes up large amounts of memory. At every generation, the simulator has to calculate the replicative fitness of pathogens based on their sequence to determine pathogen frequencies in the next generation. A naïve implementation will try to calculate the total fitness value regardless of whether the same sequence has been encountered previously. When many sequences are exact copies of each other, this brute force method wastes a significant amount of processing time.

Contagion solves this problem by splitting the representation of a pathogen into two data structures: a pathogen genotype and a pathogen node. The pathogen genotype is a data structure that hold the sequence of the pathogen while the pathogen node is a virtual representation of a pathogen that refers back to its associated genotype. When a pathogen undergoes a mutation event, the pathogen node creates a new child node to represent the new state and keep track of pathogen ancestry. When the new child node is created, the program checks the set of pathogen genotypes whether this sequence has been encountered before or is a new sequence. If it has been encountered before, the new child node is associated to an existing pathogen genotype. But if the sequence is new, a new pathogen genotype is created and is linked to the new child node. A pathogen can also recombine with other pathogens to for a recombinant genotype. When this happens, new child node is created but instead has two parents instead of one. From here, the process of evaluating whether or not to create a new genotype is the same as the mutation scenario.

Through this method, the number of times the sequence is stored is reduced significantly while also gaining the benefit of tracking pathogen lineage an ancestry. Since pathogen fitness is dependent on genotype, fitness is only calculated once for every genotype and is stored within the data structure after that. Thus, when the fitness value of the genotype is needed next time, the fitness precomputed fitness is used and fitness is not recomputed again, which significantly reduces processing time.

Contagion can fall back to a naïve implementation when simulating evolution at high rates of mutation and/or recombination. When mutation and recombination rate is high, the amount of duplicated sequences in the simulation is low and the advantage associated with this method becomes insignificant. In fact, it may even result in increased computational load because of pathogen lineage tracking. This shows that certain optimization schemes may be detrimental under certain parameter ranges.

Configuration and input formatting

Condfiguration file

Contagion uses a configuration file to set the various parameters in the simulation, point to the location of input files, and inform the program how to store data generated by the simulation. The configuration file is written in the TOML format for legibility and is divided into six different parts based on their roles in setting up the simulation: simulation, logging, intrahost models, fitness models, transmission models, and optional conditions. The comprehensive list of keywords, values, and options are listed in Appendix I.

The simulation section of the configuration file contains settings that configure the global behavior of the simulation such as the number of generations to run (`num_generations`), how many repeated simulations to perform (`num_instances`), and what simulator to use (`epidemic_model`). Note that the `num_generations` parameter indicates the number of pathogen generations to simulate, not host generations. The simulation section also specifies the paths to the pathogen sequence file and the adjacency list file to be read into the simulation. As previous discussed in *Epidemic simulators*, Contagion has many built-in simulator options based on the concept of compartmental models. Each simulator provides a different type of behavior which tries to mimic a particular type of infection and recovery profile of diseases. For instance, the SI model can be used to simulate chronic infections that lasts for a long time. For infections that do not produce long-lasting immunity, the SIS model can be used to model the reinfection of individuals that have already been infected previously. On the other hand, the SIR model can be used for infections that produces long-lasting immunity. In this model, individuals that have recovered from infection are completely removed from the infectible population. Thus, individuals can only be infected once during the span of the simulation. A detailed description and examples of the pathogen sequence file format and the adjacency list file format can be found below.

The logging section contains are the settings related to how the program writes data generated by the simulation to disk. This includes the path where to save the data log (`log_path`) as well as the frequency of logging in terms of simulation time (`log_freq`). By default, the program logs simulation data every generation. However, for large or long-running simulations, logging every generation may create enormous files in the order of hundreds of gigabytes for a single instance. In these cases, per-generation resolution may be not be necessary and the interval between records can be increased to decrease logging frequency.

An intrahost model defines the properties related to genetic simulation of pathogen populations within hosts during infection. Parameters include the mutation rate (`mutation_rate`), recombination rate (`recombination_rate`), and the rules that govern how pathogen population size changes between generations (replication_model). Currently, there are three replication models implemented in Contagion: (1) constant population, (2) Beverton-Holt, and (3) fitness-based. The constant population model sets a constant pathogen population throughout the duration of the infection. The Beverton-Holt model is an ecological model that uses a geometric progression equation to model the dynamics of population size changes based on the current population size, growth rate, and a threshold carrying capacity (Beverton & Holt, 1957). Finally, the fitness-based model assumes that the simulation uses an absolute fitness value-based (additive fitness matrix or motif model) and that population size changes are controlled by the growth rates computed from those fitness models. To prevent infinitely-large populations, the fitness-based model can set a cap to the population size. When the pathogen population within the host hits this threshold, intrahost dynamics becomes similar to the constant model. However, populations can still crash when the average growth rate fall below 1.0. Unlike the previous sections, the interhost model section can be declared multiple times to create more than one type of intrahost model in order to model groups of hosts with different intrahost dynamics, i.e. different host species. An intrahost model can be associated to a particular host using the `host_ids` parameter.

A fitness model describes the rules to calculate the replicative fitness of pathogens based on their sequence. As mentioned in *Selection* previously, Contagion has three fitness model options: (1) multiplicative, (2) additive, and (3) motif. Together with the fitness model setting, the user also needs to indicate the path to the fitness model file (`fitness_model_path`) in order for the program to import it into the simulation. Like the intrahost model, multiple fitness models can be created by declaring the section multiple times. A fitness model can be associated to a host by including the host's ID in list of ID's in the `host_ids` parameter.

Setting the transmission model is the last requirement to creating a simulation in Contagion. A transmission model indicates the transmission probability (`transmission_prob`) and transmission size (`transmission_size`) of hosts associated to this model. The transmission probability describes the chance that disease-causing pathogens will transmit given sufficient contact between hosts, represented by the presence/absence of a connection between two hosts. Transmission size specifies how many pathogens are transmitted given that transmission occurs. By default, transmission size is a constant value which may be an unrealistic condition. The mode parameter can make the number of pathogen migrants a Poisson random variable instead by setting `mode = “poisson”`.

Host population network

To input the host connections into the program, Contagion reads an adjacency list text file, and refactors it into a list of host neighbors for each host in the population. An adjacency list is a way to represent a network as a list of directly-connected pair of hosts. The format for creating this file can be found in Appendix I Figure 28 and Figure 29. In its current form, Contagion uses a static host population network that does not change throughout simulation time. However, development is currently ongoing to incorporate dynamic host connections that assemble probabilistically.

Pathogen sequences

The initial sequences and set of categorical states to be used in the simulation are given by the user using a FASTA-like format with the additional Contagion-specific style conventions (Appendix I, Figure 30). The format of the input sequence uses all the rules of the FASTA format such as using the `>` character to define the identifier line. Contagion adds two formatting rules in order to encode the sequence into the simulation and establish the relationship between available categorical states and positions in the transition matrix. First, the encoding line, a line prefixed with the `%` character, must declared prior to any sequence in order to identify to Contagion the list of categorical states to be used. In the encoding line, key-value pairs are given to show the relationship between the current character encoding and its unsigned integer representation during the simulation. The format for creating this file can be found in Appendix I Figure 30.

These integer values not only represent their respective categorical states, but also the positions of their transition probabilities in the conditioned transition rate matrix. Given that A is encoded as 0, to get the probability that A turns to T given that A will mutate, the program will refer to the first row of the transition rate matrix to get the probabilities that A turns to T, C, or G. If instead the site contains a C character, the program looks at the third row of matrix because C is encoded as 2. As a consequence of this notation, the set of categorical states must be encoded into unsigned integers starting from 0 (as the program counts from 0), incrementing by 1 without skipping. The last encoded categorical state is expected to have a value $l - 1$ if l is the total number of states.

Fitness matrix

To incorporate selection into the genetic simulation, Contagion uses a fitness matrix to compute the replicative fitness of pathogens based on their sequences. The fitness matrix is expected to be in the shape $M \times N$ where M is the number of sites in the sequence and N is the number of possible categorical states per site. If most sites share the same fitness value vector, Contagion offers a convenient way to declare a default fitness vector and only list down sites that have a different value (Appendix I, Figure 37).

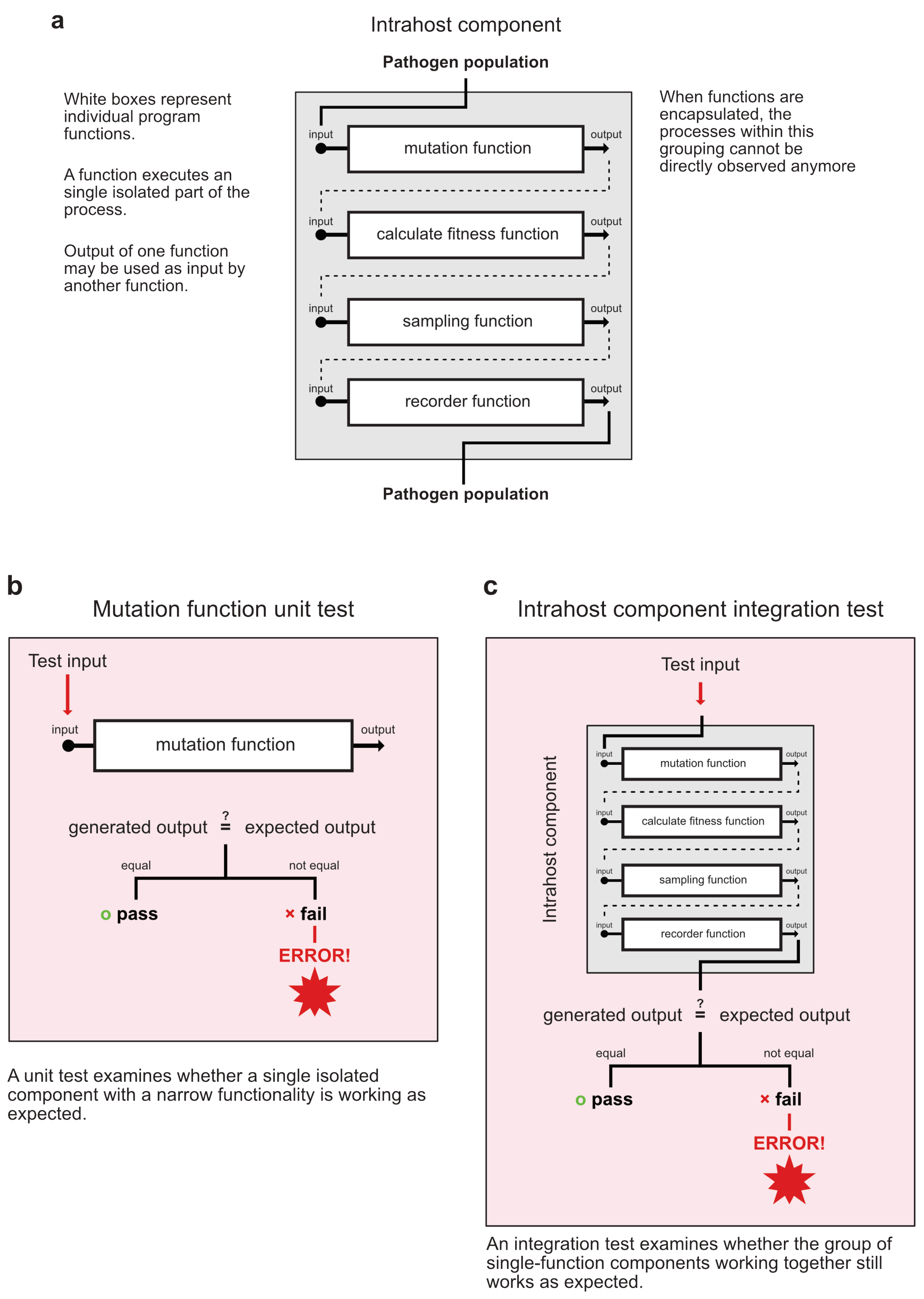
The values listed in the fitness matrix file and the selected fitness model must be consistent to prevent errors during initialization. If the fitness model was set to multiplicative, then the program expects a fitness matrix with fitness values in log space. On the other hand, if the fitness model was set to additive, Contagion expects positive values in standard base 10.

Testing and debugging

Two different testing protocols – unit tests and integration tests – were used to ensure that the program is working as intended. In software engineering, unit testing is a method that independently tests individual parts of the source code such as functions and control procedures to determine if they execute without fault, and to check whether the computed output matches an expected value (Figure 7b). This type of test ensures that each part of the program is working as expected. However, this does not guarantee that these parts will work together properly. To test the functionality of combinations of independently-working parts, integrations tests are performed. Integration tests also compares the computed output with an expected value like unit tests. However, instead of testing limited-function parts of the program, integrations testing examines whether multiple parts integrated together work as expected, hence the name “integration test” (Figure 7c).

To perform a unit test on a specific function or part of the program, a test function is created (pink box in Figure 7b) which runs the target to be tested (one of the white boxes in Figure 7a). If the target accepts input parameters and provides some kind of output, the test function will test whether a given set of inputs will result in an output equal to some expected value. If the generated output does not match the expected output, the unit test function raises an error to alert the unexpected behavior.

Because a unit test checks if the function will break for some input value, the set of inputs and expected outputs to be tested against determines the completeness of the test. It is recommended that a target is tested against an array of inputs covering the domain of expected inputs. For example, if an input parameter expects a signed integer, the target should be tested with very small value negative integers, very large positive integers, integers close to zero, and zero itself. The unit test function also catches any unexpected errors raise by the target. For example, if the mutation function suddenly encounters an error while mutating a site, the unit test function halts testing and will immediately raise an error to record that something unexpected occurred. Thus, unit testing helps test whether the code is actually working, and if the programmed behavior matches expectation.



White boxes represent individual program functions.

A function executes an single isolated part of the process.

Output of one function may be used as input by another function.

When functions are encapsulated, the processes within this grouping cannot be directly observed anymore

A unit test examines whether a single isolated component with a narrow functionality is working as expected.

An integration test examines whether the group of single-function components working together still works as expected.

Figure 7. Software testing procedure diagram.

The set of calculations necessary to perform intrahost pathogen evolution in (a) is used as an example to show the similarities and differences between unit testing (b) and integration testing (c). In this diagram, white boxes represent functions, statements, or any part of the program with limited or singular functionality. Gray boxes indicate encapsulation or grouping together of functions with related and interdependent functionality. Pink boxes represent test functions.

If a module passes its unit test, then that part of the program is guaranteed to work as expected for the given parameter range used to test it. However, this type of test does not guarantee that the function will work for all types of parameter values, nor does it guarantee that the target when working in combination with other parts of the program will also run as expected. Integration tests address this issue and tests whether the aggregation of individual targets also work as expected (Figure 7c).

To perform integration testing, another test function is created (pink box in Figure 7c) which runs the target to be tested (the gray box which encapsulates the white boxes, Figure 7a). Unlike the unit test, an integration test deals with more complex functionality that is often composed of multiple interdependent parts. Because of this, integration tests are performed only after all parts have passed their respective unit tests. This helps narrow down the problem to improper integration between parts since each part has been shown to work properly on its own. Integration testing also uses a set of input parameters and some corresponding expected output to compare against. Integration testing assumes there is some way to reliably predict or analytically compute the true answer for a complex set of actions. Thus, if the behavior of the integrated system is unknown or unpredictable, then integration testing cannot be done.

For Contagion, I created test functions that check the major parts of the code such as the random sampling functions for replication, mutation, and spreading of the pathogen; and fitness computation that calculate the overall fitness value of each pathogen. To perform integration tests, I applied a bottom-up approach given the hierarchical nature of the program's design. A bottom-up approach first tests the lowest level components using unit tests, then tests aggregations of these components. This process if repeated until all lower-level components are integrated and testing reaches the top of the hierarchy. The parameters and source code for unit test and integration test functions can be found online at https://github.com/kentwait/contagion.

RESULTS

The SIS individual-based network epidemiological model

I simulated the spreading of a disease that follows the SIS compartmental over a complete network to determine if a network version of the SIS model reaches equilibrium or is prone to chaotic behavior. A hallmark of the standard SIS model is its ability to reach equilibrium at 0 and a positive number such that the number of infected individuals remains constant at equilibrium. In the discrete version of the SIS model, this positive equilibrium is not guaranteed for all parameter values. While Contagion runs simulations in discrete time, it also uses the concept of individual-based modeling and networks. In this approach, each individual in the simulation is an autonomous agent that acts based on inputs from the environment as well as its internal state. Moreover, actions involving other individuals in the system is permitted only when a connection between individuals exists. Given these constraints, it is unclear how the network model will behave.

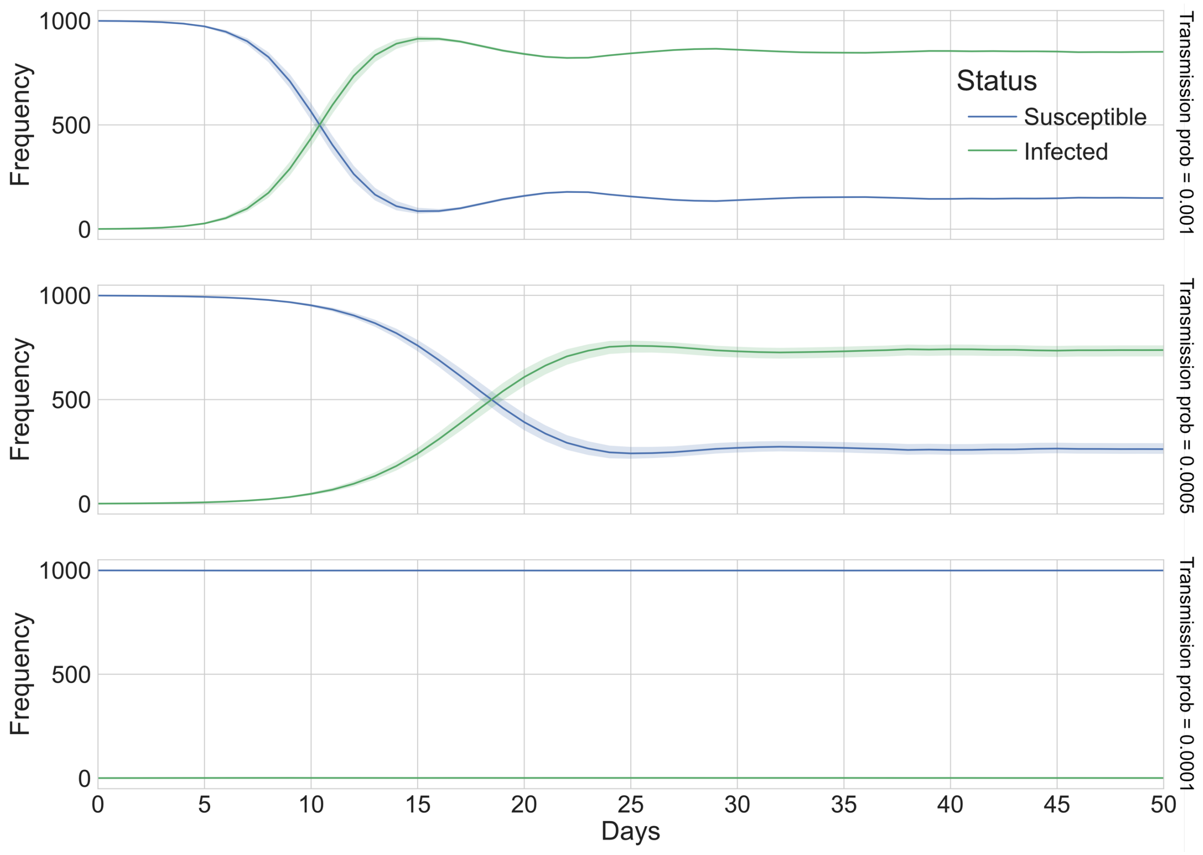


Figure 8. Frequency of susceptible and infected individuals over time in the Network SIS model.

Time-series plot of the number of susceptible (blue) and infected (green) individuals in a host population of 1000 individuals connected via a complete network. The first time-series plot (top) shows the frequency of individual statuses when per-connection transmission probability is 0.001 transmissions/connection/day. The second (middle) and third (bottom) time-series plots show the change in frequencies over time for 0.0005 and 0.0001 transmissions/connection/day respectively. The solid line indicates the mean frequency for at each time point while the highlight shows the 95% CI.

Repeated simulations of an SIS-type disease showed that the network model initially show chaotic behavior such that the frequency of infected individuals oscillates around a certain value. However, the amplitude of the oscillation diminishes over time and the system can reach a stable equilibrium. High transmission rates appear to create larger initial oscillations compared to smaller values. Since the transmission rate in Contagion describes the probability of transmission for every connection to a susceptible neighbor, the actual number of new infections at every time step is a binomial random variable. This is a key distinction that makes the network SIS model in Contagion stochastic, as opposed to the deterministic behavior of compartmental models.

In models that do not possess a feedback loop, such as the SIR model, chaotic behavior is not expected to occur. In the SIR model, individuals traverse the susceptible, infected and removed compartments at most once such that individuals can never become susceptible again. Therefore, there is no reason to expect that the network version of the SIR model will show chaotic behavior.

Simulations of the network SIR model an interesting departure from both the continuous-time and discrete-time SIR models (Figure 9). In standard SIR models, the shape of the infected curve is usually skewed such that the right-hand side has a longer tail than the left-hand side. This pattern occurs because the length of time an individual is infected is specified by a recovery rate parameter instead of a time interval (Equation 1). Although the reciprocal of the recovery rate γ gives the average duration of the infection, the rate indicates the probability that an infected individual will no longer be infected at for every time unit such that infected individual can recover earlier or later than the expected duration. Although the reciprocal of the recovery rate γ gives the average duration of the infection, setting a constant value equal to the average gives a completely different dynamic because there is no variance in the duration of infection. Setting the duration of infection means the waiting time until moving to the removed compartment now constant and the infectious duration is no longer Markovian. Thus, the recovery times of infected individuals depends on the time they enter the infected state. This explains why the infected curve in the simulation is symmetric whereas the standard SIR models produce a right-skewed curve.

The SIR-like individual-based network epidemiological model

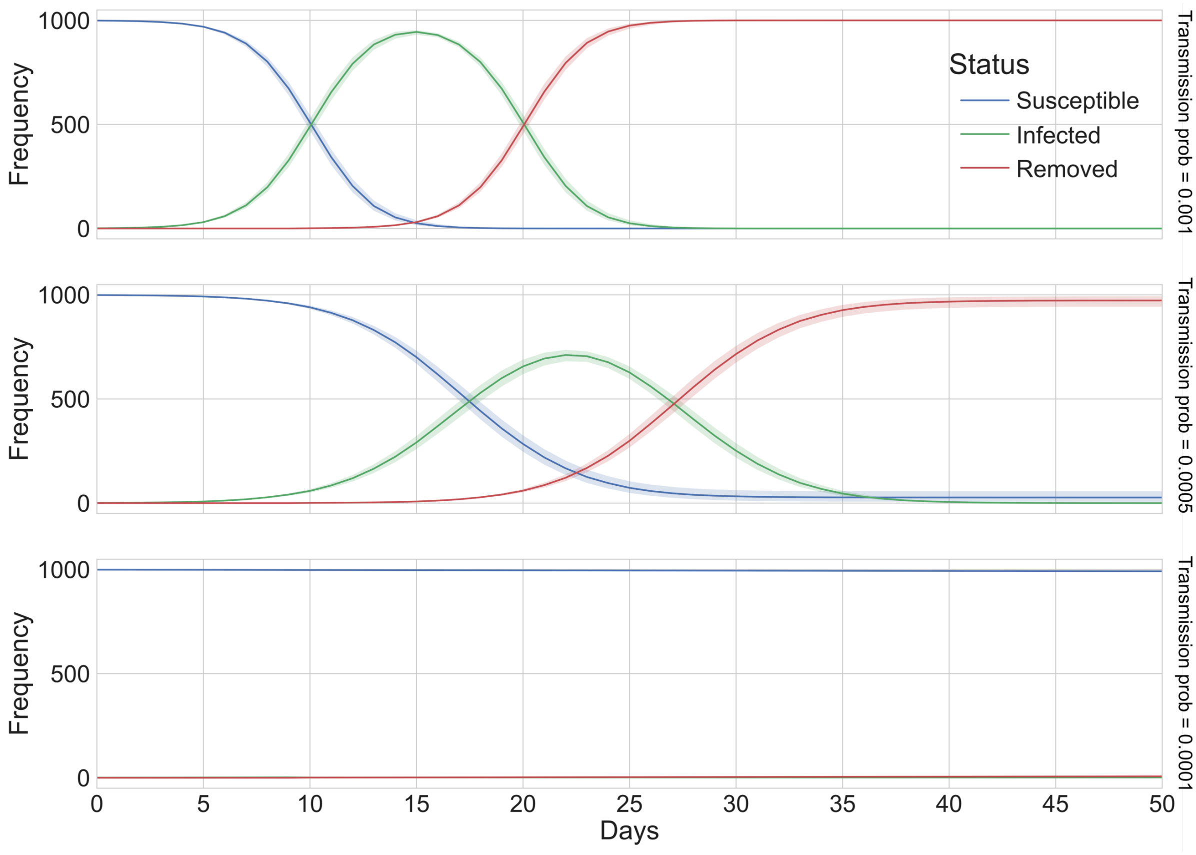


Figure 9. Frequency of susceptible, infected, and removed individuals over time in the Network SIR* model.

Time-series plot of the number of susceptible (blue), infected (green), and removed (red) individuals in a host population of 1000 individuals connected via a complete network. The first time-series plot (top) shows the change in frequencies over time when per-connection transmission probability is 0.001 transmissions/connection/day. The second (middle) and third (bottom) time-series plots show the change in frequencies over time for 0.0005 and 0.0001 transmissions/connection/day respectively. The solid line indicates the mean frequency for at each time point while the highlight shows the 95% CI.

It is arguable whether setting an infectious period instead of a recovery rate is incorrect or not. There are merits to the a constant or normally-distributed infection time. Studies have shown that adopting a constant time interval, instead of an exponentially distributed waiting time, can lead to derived from empirical observations more realistic prediction (Lloyd, 2001; Vergu et al., 2010). In this case, a constant infection time is advantageous because Contagion uses the time a host is infected to simulate pathogen molecular evolution within each infected hosts. Having an exponentially-distributed infection time would mean that number of pathogen generations will also be exponentially-distributed.

Fixation probability of mutations within a single host

To validate the genetic simulator that facilitates intrahost evolution during host infections, I simulated the fixation of mutants starting from an initial frequency p and determined the fixation probability at different values of the scaled selection coefficient Ns. I chose this method to validate the program because the theoretical value can be easily calculated, and the fixation process tests the replication and selection processes of the simulation. I calculated the fixation probability by repeatedly running the program 10000 times and counting the number of instances the allele reached fixation. I resampled the generated data 1000 times to get the mean and generate the 95%CI of the sample.

Table 1. Comparison between theoretical and empirically-calculated fixation probabilities

Ns	***s***	***p***	**Theoretical *u***	**Mean**	**95%CI**
0	0.000	0.1	0.0000	0.1026	(0.094, 0.111)
		0.5	0.5000	0.5010	(0.472, 0.530)
1	0.002	0.1	0.2096	0.1986	(0.186, 0.210)
		0.5	0.7311	0.7320	(0.704, 0.758)
2	0.004	0.1	0.3358	0.3422	(0.329, 0.354)
		0.5	0.8808	0.8720	(0.850, 0.893)
4	0.008	0.1	0.5509	0.5358	(0.524, 0.551)
		0.5	0.9820	0.9860	(0.979, 0.993)
10	0.020	0.1	0.8647	0.8712	(0.861, 0.881)

	0.5	1.0000	1.0000	(1.000, 1.000)

The program's results did not significantly deviate from the theoretical predictions for the range of scaled selection coefficients that I tested Table 1. The theoretical values were consistently within the 95%CI for all parameter combinations used. These results show that the genetic simulator aspect of the program is working as expected.

Pathogen sequence mutation rate

Table 2. Expected number of mutations after one generation.

Mutation rate μ (site^{-1} gen^{-1})	**Population size N**	**Number of sites l**	**Mean (site^{-1} gen^{-1})**
1×10^{-6}	1000	10000	0.98×10^{-6}
	10000	1000	0.99×10^{-6}
	100000	100	1.02×10^{-6}
1×10^{-4}	100	10000	0.99×10^{-4}
	1000	1000	1.00×10^{-4}
	10000	100	1.01×10^{-4}
1×10^{-2}	100	10000	1.00×10^{-2}
	1000	1000	1.00×10^{-2}
	10000	100	1.00×10^{-2}

Finally, I tested whether the program was producing the expected amount of mutations. To test the program's mutational process, I created a population of N pathogens with 10000 sites under a 2-allele model and gave all pathogens the same initial sequence. Given the per-site per-generation mutation rate, I let the program perform one round of mutation on the entire population of pathogens and counted the number of mutated sites after. This process was performed using different mutation rate values and each set-up was repeated 100 times to get the expected mutation rate per site per generation. Results showed that average mutation rate was close to the expected value and within the 95%CI.

Although it is impossible to prove that program is without bugs or defect, I have shown through these tests that the genetic simulator behaves according to theoretical predictions and the epidemic simulator does not suffer from chaotic behavior and works in a predictable manner, with slight deviations from the standard compartmental models. Aside from these tests, the functions and methods in the programs are tested individually using unit tests to ensure that each function works as intended.

DISCUSSION

Contagion is a new simulation program that models the state and evolution of information that is being transmitted across a network. To achieve this, the program concurrently performs two processes – evolving information states present at each vertex in the network and choosing which pieces of information to transmit and to which nodes to transmit it to. During the simulation, the program tracks which host individuals have received information, are transmitting information, and have yet to receive any information. Contagion also records the frequencies and types of information present within each host and over all individuals at particular intervals. Whenever the content of the information mutates, the program also stores the time and location of emergence, and records from which piece of information it arose from.

Any type of information that can be encoded as a sequence of categorical states can be used in Contagion. For example, biomolecules such as DNA, RNA, and proteins are polymeric sequences composed of categorical states such as nucleotide bases adenine (A), thymine (T), cytosine (C), guanine (G), and uracil (U) in nucleic acids, and the 20 amino acids in proteins. Contagion is not limited to biological sequences. Language can also be modeled in Contagion by setting words as a series of discrete categories and phrases as a sequence of these words. Therefore, any type of information that can be treated as a categorical series can be simulated in the program.

The novelty of the program lies in the ability to combine processes occurring at two different scales. In the case of pathogens, Contagion is a useful tool to study the effects and interactions between within- and between host process processes. Although previous studies have constructed software with similar goals as Contagion (Leventhal et al., 2015; Park, Loverdo, Schreiber, & Lloyd-Smith, 2013; Read & Keeling, 2006), these programs were specifically designed for a particular scenario. To my knowledge, Contagion is the first general purpose simulation software that delivers this kind of functionality. I hope that this piece of software enables more studies in the field to better understand pathogen evolution.

CHAPTER 3
HOST NETWORK TOPOLOGY AFFECTS THE SPREAD OF NEW MUTATIONS

INTRODUCTION

In epidemics, it is frequently observed that a few individuals appear to be responsible for spreading the disease to a disproportionate number of people. This interesting observation has driven epidemiological studies to examine the role of networks on the spread of infection. In cases where "superspreading" events have been observed, this is often a key moment in the epidemic that explosively increases the incidence of the disease (Fujie & Odagaki, 2007; Yu et al., 2007). Unfortunately, these dynamics cannot be readily examined though standard epidemiological models which assume a well-mixed population. To address the shortcomings of existing epidemic models, epidemiological studies have incorporated network theory and individual-based models to include more realism in their models (Meyers et al., 2005; Starnini, Machens, Cattuto, Barrat, & Pastor-Satorras, 2013; Stolerman, Coombs, & Boatto, 2015). Simulation of epidemics on different networks have showed that the uneven distribution of connections in the host population is a key factor affecting the spread of infections (Leventhal et al., 2015; Lloyd-Smith, Schreiber, Kopp, & Getz, 2005).

The distinguishing characteristic of pathogen evolution lies in the necessity of pathogens such as viruses, to infect host organisms in order to reproduce and avoid extinction. Yet this behavior is not currently captured in existing population genetics models (Lloyd-Smith, Funk, McLean, Riley, & Wood, 2015). During infection, new mutations may appear following replication and change in frequency within the host either through random genetic drift or by conferring a competitive advantage or disadvantage to the mutant. However, new genotypes that appear within the host are confined within the host unless the mutant is transmitted and infects other hosts. Thus, the distribution and frequency of mutations is affected not only by within-host factors such as mutation rate, population size, and strength of selection, but also by between-host factors like transmission rate, connectivity, and network topology.

Traditional population genetics models do not account for this pathogen-specific demography because of the simplifying assumption that populations are well-mixed and unstructured. While this an appropriate null model for free-living organisms, its application as a null model for pathogens seems unsuitable. Deviation of population genetics null models, such the standard neutral model, are often cited as indication of the presence of selection (Andolfatto & Przeworski, 2000; Bhatt, Katzourakis, & Pybus, 2010; McDonald & Kreitman, 1991), or demographic changes (Haipeng Li & Stephan, 2006; Nielsen et al., 2009; Strimmer & Pybus, 2001). However, it is unclear whether unique life history of pathogens and its network-like structure significantly deviate from unstructured models, and if so by how much? Metapopulation models that explicitly consider subdivisions in the overall population are candidates to better model pathogen evolution. In this study, I showed how networks create population structure that can significantly change the behavior of the system compared to an unstructured population and demonstrate that different network topologies can significantly affect the pathogen genetic evolution by influencing the frequency spectrum of mutations, expected genetic diversity, and the fixation of mutations.

METHODS

The network SIS epidemiological model

The susceptible-infected-susceptible epidemic model is a type of compartmental model that describes the dynamics of an epidemic in which long-lasting immunity does not occur upon recovery from infection. As a consequence, previously infected individuals may be infected by the disease multiple times. The standard SIS model is composed of three ordinary differential equations that describe the number of susceptible (S) and infected individuals (I) in continuous time (Equation 9). This model assumes that susceptible and infected individuals are homogenously-mixing such that the law of mass action applies (Matthew James. Keeling & Rohani, 2007). Under this assumption, the population is free of any condition that may produce population structure.

Equation 9. SIS model

$$\frac{dS}{dt} = \frac{-\beta SI}{N} + \gamma I \tag{9.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I \tag{9.2}$$

In this model, β is the rate of new infections and γ is recovery rate. The values of S and I indicate the number of susceptible and infected individuals are related to each other by the equation $S + I = N$, where N is the total number of hosts in the population. The basic reproduction number is given by $R_0 = \beta/\gamma$. When $R_0 < 1$, then $I = 0$ and the epidemic dies out due to the lack of infectious hosts. Whereas $R_0 > 1$, the epidemic achieves a stable endemic equilibrium at $I = N(1 - \beta/\gamma)$ and is expected to persist in the population.

To transform this model into discrete time, time can be divided into time intervals Δt, and the ordinary differential equations can be reformulated into a set of difference equations (Equation 10) as shown by Allen (1994).

Equation 10. Discrete-time SIS model

$$S_{n+1} = S_n - \frac{\beta}{N}\Delta t S_n I_n + \gamma \Delta t I_n \tag{12.1}$$

$$I_{n+1} = I_n + \frac{\beta}{N}\Delta t S_n I_n - \gamma \Delta t I_n \tag{12.2}$$

(Allen, 1994)

In this discrete-time transformation, frequencies are separated by unit time Δt and time n is equal to the time at $n\Delta t$. The infection rate in the discrete-time transformation is defined by the contact rate β, which is the expected number of individuals an infectious host has had sufficient contact with in order to transmit the infection (Allen, 1994). To ensure that all solutions of these equations are positive, the following inequalities must be satisfied (Equation 11).

Equation 11. Conditions to keep the discrete-time SIS model valid

$$S_0 > 0 \tag{13.1}$$

$$I_0 > 0 \tag{13.2}$$

$$S_0 + I_0 = N \tag{13.3}$$

$$\gamma \Delta t \leq 1 \tag{13.4}$$

$$\beta \Delta t < \left(1 + \sqrt{\gamma \Delta t}\right)^2 \tag{13.5}$$

(Allen, 1994)

The basic reproduction number defines the rate of secondary infections created in the population. In the discrete-time model, the basic reproduction number is also given by $R_0 = \beta/\gamma$ (Allen, 1994). Similar to the continuous-time SIS model, when $R_0 < 1$, the disease dies out and the system reaches equilibrium at $I = 0$. However, when $R_0 > 1$ the discrete-time model is not guaranteed to reach an endemic equilibrium and is susceptible to chaotic behavior at certain parameter values.

To test the effect of network topology, I considered the discrete-time SIS model as the null model because it models the epidemic assuming an unstructured and homogenous population of hosts.

On the other hand, networks implicitly create population structure unless it is a complete graph (Newman, 2010). Population structure emerges from networks due to the limited number of connections between nodes. As a result, population structure grows stronger as the network density – the relationship between the number of nodes to the number of connections present in the network – decreases and the network becomes more sparsely connected . Network density is calculated based on the number of nodes N present in the graph (host population size) and the total number of undirected edges $\tilde{d}$ present (Equation 12.1) or directed edges $\tilde{d}_d$ (Equation 12.2). This quantity is a summary statistic that describe the average connectedness of the network.

Equation 12. Network density for undirected and directed graphs

$$D = \frac{2\tilde{d}}{N(N-1)} \tag{12.1}$$

$$D = \frac{\tilde{d}_d}{N(N-1)} \tag{12.2}$$

(Hagberg, Schult, & Swart, 2008)

I used Contagion (Kawashima, 2017) to create an SIS model embedded in a network structure. In this model, the nodes on the network represent the host population and the connections represent contact between hosts that may transmit the infection. In the simulation, infected hosts spread the disease by transmitting disease-causing pathogens to directly-connected susceptible neighbors. The probability of disease transmission is given by the following equation (Equation 13), where P_T is the transmission probability given a connection, and λ_d is the expected number of pathogens transmitted given a transmission event occurs, and k is the actual number of pathogens transmitted. This is equivalent to the probability that at least one disease-causing pathogen is transmitted to a susceptible neighbor.

Equation 13. Probability that at least one disease-causing pathogen is transmitted

$$P(k > 0) = P_T \prod_{k=1}^{K} \frac{{\lambda_m}^k \mathrm{e}^{-\lambda_m}}{k!} = P_T\left[1 - \mathrm{e}^{-\lambda_d}\right] \tag{13}$$

Thus, the contact rate β in the discrete-time SIS model is comparable to the transmission probability when a connection between source and recipient exists. When a transmission event is permitted to occur, the number of pathogens to be transmitted is determined stochastically by the program and is a Poisson random variable with an expected value of λ_d. Given $\lambda_d > 0$ and the within-host pathogen population size $N_d \geq \lambda_d$, pathogens are randomly sampled from the within-host population.

In some cases, multiple infections or coinfections of different pathogen strains from different infected sources can occur (Seabloom et al., 2015; Susi, Barrès, Vale, & Laine, 2015). Coinfection describes the phenomenon where multiple strains or types of pathogens simultaneously infect a single host. If coinfections are not allowed, infected neighbors of infected hosts block the transmission of the infection and creates a similar dynamic to removed individuals in the SIR model. If coinfections are allowed, infected individuals can transmit pathogens both to susceptible and infected neighbors. In this case, susceptible hosts move into the infected compartment, but already infected hosts remain infected and does not reset the interval of infection. A side-effect of coinfection is that it increases the mixing of the overall pathogen population and may influence genetic diversity and fixation dynamics (Susi et al., 2015).

In all network SIS simulations in this study, transmission probability per connection is $P_T = 1.0$, the number of pathogens transmitted per connection is stochastically determined and is given by the expected number of migrants $\lambda_m = 5$, and the duration of infection $1/\gamma = 10$, unless otherwise stated. This simplifies the stochastic transmission simulations by regulating it with a single random variable.

Selection model for studying pathogen genetic evolution

Natural selection of pathogens is based on the relative replicative fitness of each pathogen computed from their genotype using a given multiplicative fitness matrix. In the multiplicative fitness matrix model, each categorical state per site is assigned a fitness contribution f_i and the product of all the fitness contributions becomes the relative replicative fitness of the pathogen F (Equation 14.1). To get the frequencies of pathogen genotypes in the next generation, the list of replicative fitness values of the set of genotypes present in population is normalized such that they sum to 1.0 (Equation 14.2). Once normalized, the elements of vector $\widehat{\mathrm{F}}'$ specify the probabilities of classes of a multinomial distribution. Given a constant within-host population size N_d, the frequencies of genotypes in the next generation X_1 ... X_k, where k represents the genotype, are decided by repeatedly sampling the multinomial distribution N_d times such that the offspring picks its parental genotype. With a multinomial distribution whose probability mass function is given by (Equation 14.3), the frequencies of genotypes in the next generation X_1 ... X_k are non-negative integers $\mathrm{x}_1 \ldots \mathrm{x}_\mathrm{k}$ such that $\mathrm{X}_1 = \mathrm{x}_1 \ldots \mathrm{X}_\mathrm{k} = \mathrm{x}_\mathrm{k}$.

Equation 14. Multiplicative fitness model

$$F_k = \prod_{i}^{I} f_i \text{ for each } k^{th} \text{ genotype} \tag{14.1}$$

$$\widehat{F}' = \frac{F_k}{\sum_k F_k} \tag{14.2}$$

$$Pr(X_1 = x_1 \ldots X_k = x_k) = \frac{n!}{x_1! \ldots x_k!} F'_1 \times \ldots \times F'_k \tag{14.3}$$

When selection is present, it affects the frequency of pathogen genotypes in the within-host population but does not affect the probability of sampling pathogen for transmission. If the number of pathogens to be transmitted $k > 0$, pathogens are sampled randomly from the within-host population such that the probability of sampling a particular pathogen genotype is frequency-dependent.

Network generation

I considered three different network topologies – regular, binomial, and scale-free – to determine if network effects affect the evolution of pathogens. To construct these networks, I used the random graph constructors in the NetworkX Python package (Hagberg et al., 2008).

To create the regular network, I used the `networkx.random_regular_graph(`d`, `N`)` constructor method where the number of degrees for each node is $d = 5$ or $d = 10$, and the number of nodes in the network $N = 200$. The random graph is constructed according to the Steger and Wormald's algorithm (1999). This graph constructor produces a d-regular graph with a random set of connections to nodes and guarantees that all nodes have d number of connections.

For the binomial network, I used the `networkx.fast_gnp_random_graph(`N`, `P_c`)` constructor method (Batagelj & Brandes, 2005) where the number of nodes $N = 200$, and the probability of edge creation is $P_C = 0.025$ or $P_C = 0.050$. Note that for this kind of graph generator, neither the number of edges per node nor the total number of edges in the graph can be specified beforehand. Thus, in order to get an average degree matching $d = 5$ and $d = 10$, probabilities $P_C = 0.025$ or $P_C = 0.050$ were selected respectively. The Batagelj and Brandes algorithm is based on the Erdos-Renyi "$G(n, p)$" model (1959) with optimizations to make graph construction more efficient. In the Erdos-Renyi model, a binomial graph is constructed by starting with a disconnected network that consists of N nodes and no edges. Then, edges are added by testing all pairs of nodes one by one independently with a probability of edge creation P_C (Erdős & Rényi, 1959).

Finally, for the scale-free network, I generated the network using the `networkx.powerlaw_cluster_graph(`N`, `d_r`, `P_g`)` constructor method (Holme & Kim, 2002) such that the number of nodes $N = 200$, the number of random edges to add for every new node $d_r = 5$ or $d_r = 10$, and the probability of creating a triangle subgraph after adding a random edge $P_A = 0.9$. To construct the scale-free network, the Holme and Kim algorithm starts with a network that consists of N nodes and no edges and iteratively adds new edges. At every iteration, one node is selected in lexicographic order and d_r edges are added. For each new d_r edge, it is attached to one of the existing nodes at a probability proportional to node's number of existing connections. This creates a positive feedback loop where nodes with a large number of connections are more likely to accrue even more new connections during graph construction. After this preferential-attachment step, the Holme and Kim algorithm randomly adds another edge between the current origin node to any one of the recipient node's connected neighbors with probability P_A to create a triad. If all neighbors of the recipient node are already connected to the origin node, then preferential attachment is performed instead.

Since these graph constructors create random graphs, the number of edges, network density and clustering cannot be determined *a priori*. Moreover, random generation can produce graphs with singleton nodes and disjointed subgraphs. To address these concerns, I created a sample of 1000 randomly-generated networks per constructor type to determine the mean and variance of each network's density and average per-node clustering. Then, I generated the respective networks until it closely matched the mean values of these network measurements. This process ensured that the networks I used are close the expected graph for the given set of graph generator parameter values.

Apart from changes to the degree distribution, the clustering of nodes in the network may also be affected by a particular topology. To measure the tendency of nodes to group together, I used the `networkx.clustering` method to compute the distribution of clustering coefficients. Clustering coefficient is a per-node quantity that describes the tendency of nodes to group together. Formally, for unweighted or uniformly weighted graphs, the clustering coefficient of a node x is the fraction of triads or triangular connections that exist that passes through node x. In Equation 15, the clustering coefficient c_x of node x is calculated using the number of triangles through node x $A(x)$ and the node's degree $d(x)$. Note that self-loops or connections originating and ending at the same node are ignored.

Equation 15. Clustering coefficient of a node

$$c_x = \frac{2A(x)}{d(x)(d(x)-1)} \tag{15}$$

After constructing these networks, I generated their adjacency list representation and imported it into Contagion. Below are the networks I used for to test the influence of networks on fixation and genetic diversity (Figure 10).

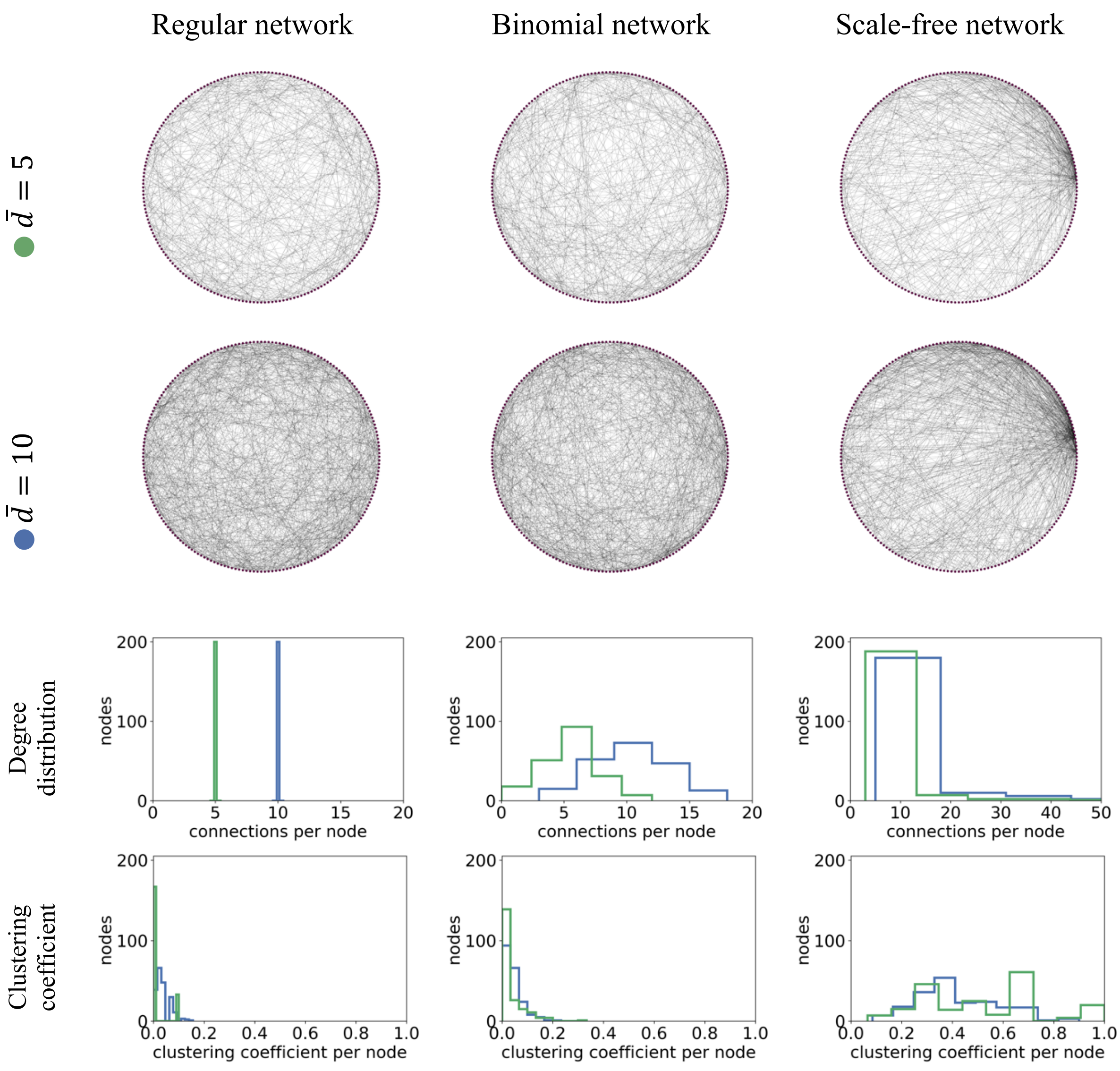


Figure 10. Networks used in this study.

Regular, binomial and scale-free random networks were used to model the underlying contact network in the host population with network densities $\boldsymbol{D = 0.025}$ *(top) and* $\boldsymbol{D = 0.050}$ *(second row) such that the average number of connections per host is equal to* $\boldsymbol{\bar{d} = 5}$ *and* $\boldsymbol{\bar{d} = 10}$ *respectively. The distribution and configuration of the connections are illustrated by the density of edges in the circular diagram and the degree distribution histogram (third row). The distribution of clustering coefficients is also shown (bottom row).*

Fixation probability of a mutation spreading over a network

To test the effect of different network topologies on the fixation of mutations, I repeatedly simulated the spreading of pathogens over different network topologies and measured the number of fixation events that occurred. I considered a scenario where pathogens have two genotypes, U and P. Initially, the frequency of U and P were set to $p = 0.5$ for all infected hosts at the start of the simulation. Then, I created two different initializations to control for the effect of the number of initially-infected hosts – at equilibrium or from a single infected host. In both scenarios, initially-infected infected hosts were randomly assigned for every realization of the simulation. However, for the equilibrium scenario, the assignment is weighted to mimic true equilibrium where infected hosts are more likely to be found in high-degree hosts than low-degree hosts. Mutation was not allowed during the simulation in order to ensure that any change in genotype frequency was the result of selection, drift, transmission, or network effects. For each combination of network and parameters, the simulation was run until the one genotype reached fixation. If the simulation does not reach fixation after 100,000 pathogen generations, the current realization is aborted and dropped. Each simulation was repeated for 1,000 instances to determine the probability of fixation. The 95% confidence interval for each dataset was generated by bootstrapping the simulated outcomes 1,000 times.

The fixation probability within a network was also compared to the theoretical value that assumes a homogenously-mixing population of pathogens using the equation derived by Kimura and Ohta (1969) adapted for haploid populations (Equation 16).

Equation 16. Fixation probability of a mutation given infinite time.

$$u(p) = \frac{1 - \mathrm{e}^{-2Nsp}}{1 - \mathrm{e}^{-2Ns}} \tag{16}$$

For all simulations, $N = N_d = 500$, $p = 0.5$, and s is the selection coefficient within individual hosts, unless otherwise stated.

Site-frequency spectrum of pathogens spreading over a network

The site frequency spectrum, also known as the allele frequency spectrum, is the distribution of allele or variant frequencies over a set of sites in a sample of sequences. The site frequency spectrum summarizes the distribution of allele frequencies as a histogram depending on the number of sampled sequences, or as binned values. To construct the site frequency spectrum, each site (column) of a multiple sequence alignment is inspected and classified according to the derived allele frequency in the current locus. This corresponds to one entry in the frequency spectrum. As more sites along the alignment are classified, the site frequency spectrum becomes a tally of how many times a particular derived allele frequency was observed. By getting the total number of times each derived allele frequency class is observed, the site frequency spectrum can describe evolutionary forces acting on the population from which the samples were taken from. For example, if there is an overabundance sites with high derived allele frequency, then these sites could be being driven to high frequencies by positive selection.

There are two implicit assumptions in interpreting frequency spectrums. First, sites are independent of each other such that changes in allele frequency of a particular site do not influence the allele frequencies in other sites (Xie, 2011). When sites are linked, it becomes difficult to determine whether the change in frequency is caused by this particular derived allele's effect or due to the changes occurring in other sites. In a complete-linked case, the probability of observing an allele at any site becomes dependent on the total fitness of all the sites considered. Because all sites are linked, a mutation at a particular site increases frequency when then the entire haplotype increases in frequency, or by high mutation rates producing the same mutation at the same site but in a different background. If all occurring mutations do not change the fitness of the pathogen, then the expected frequency of sites from low to high derived allele frequency decreases geometrically as the chance of observing high derived allele frequencies becomes dependent on the probability that the site experiences multiple independent mutations. Another implicit assumption in the site frequency spectrum is that there are exactly two alleles present for every site in the sequence such that one is ancestral and the other is derived, but multiallelic methods have been developed as well (Long, Williams, & Urbanek, 1995; Zeng, 2010).

Since pathogens such as RNA viruses have been shown to experience low recombination (Chare & Holmes, 2006; Rico-hesse, 2003), their genetic sequence can be considered as a completely-linked haplotype block. As a consequence, while the site frequency spectrum can be constructed, it cannot be directly compared to the Poisson random field model that assumes free recombination (Sawyer & Hartl, 1992) and the standard neutral model is an inappropriate null model. To overcome this problem, I considered a completely-linked sequence under constant population size (or mimicking the population size changes of the alternative model) in an unstructured population as the null hypothesis. If network structure does not significantly change the dynamics of evolution, then frequency patterns should be consistent with the null model. Given that this is a completely-linked case, I also consider the average pairwise difference between haplotypes as a measure of genetic evolution.

To construct the site frequency spectrum, I randomly sampled 100 simulated viral pathogens after 1000 generation. Pathogens in the network are regarded as a single pool such that sampling is performed without regard for their host or position in the network. Then, a multiple sequence alignment is constructed from the sample of 100 pathogen sequencing. Each pathogen has a genome that is 1000 sites long, and at each site containing either a U or P character. Each site (column) in the multiple sequence alignment is classified by comparing the current state of the site to the ancestral state, counting the number of samples possessing the derived allele (state), and recording the frequency of derived allele for the particular site. This process is repeated for each site in the sequence until all sites have been classified according to the number of derived alleles present. From the list of derived states frequencies, a histogram is constructed to count the number of sites belonging to each derived state frequency class from 0 derived states (no sample has the derived allele) to 100 derived states (all samples have the derived allele). The site frequency spectrum includes derived frequency classes from 1 allele (singleton) to 100 alleles (fixed) and excludes the 0^{th} frequency class.

RESULTS

SIS epidemic spreading over a connected host population network

The susceptible-infected-susceptible (SIS) epidemic model is a compartmental model that assigns hosts into one of two compartments or states – susceptible and infected. In this model, infected individuals revert back to being susceptible to infection after a given period or at a particular rate. The ability of the SIS model to reach a non-zero equilibrium was a key reason for choosing the model to examine the effects of networks on pathogen evolution. Figure 11 shows the change in the number of susceptible and infected hosts over time for different network topologies oscillates around 180 infected individuals in a network of 200-connected hosts.

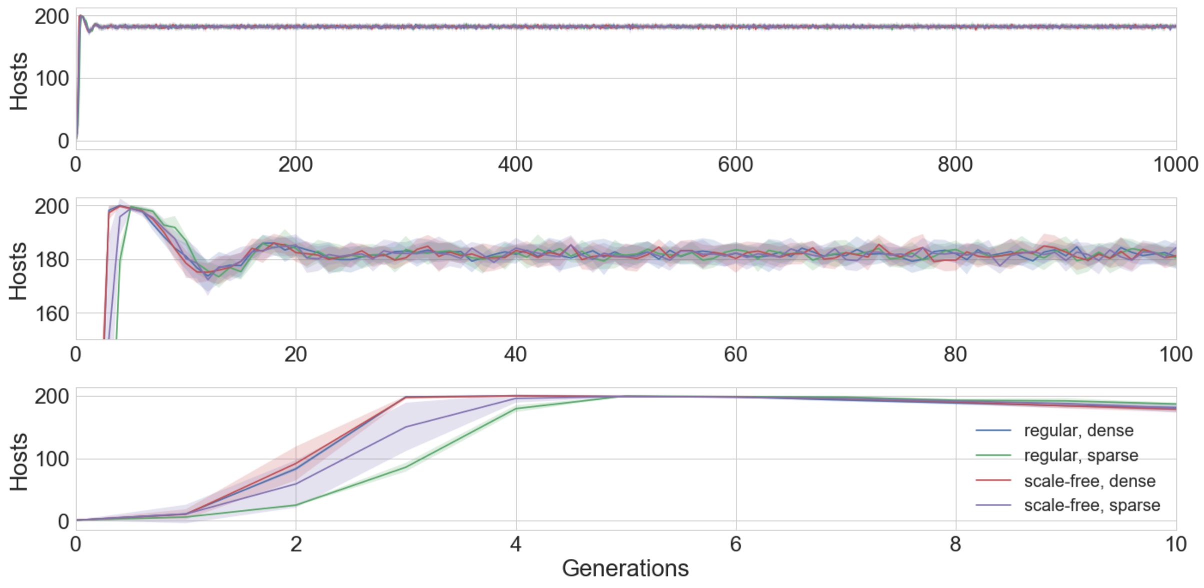


Figure 11. Frequencies of infected hosts over in SIS simulation on regular and scale-free networks.

Number of infected hosts at each time point in the simulation ($1/\gamma=10$) over generation time (pathogen generations) for the entire simulation is shown (top). Magnified views for the first 10 (bottom) and 100 generations (middle) are also shown to observe the contrast between the different network topologies and network densities.

The speed of the initial outbreak distinguishes the underlying host population network. Here I compared the effects of the density of connections and the distribution of connections in the network. While all versions of the simulation reached equilibrium, results show that the more connected networks (regular, dense and scale-free, dense) were slightly faster on average than networks with half their density (regular, sparse and scale-free, sparse) (Figure 11). Notice that the speed of the outbreak was extremely rapid, which went from a single infected individual to infecting close to, or even all hosts in the population.

The density of connections describes the number of connections per host d in relation to the total number of hosts present N (Equation 12). Imagine a homogenously-mixed population where each host has the ability to encounter and come into contact with every other host. This model describes a host population connected through a "complete network" where each host is connected to every other host and has a network density $D = 1$. This network model is equivalent to the unstructured population if and only if migration is unconstrained. As a result, interactions between hosts in the complete network also follows mass action kinetics. Therefore, the complete network serves as the upper limit in terms of the number of connection hosts can have such that all other networks are more sparsely connected ($D < 1$) than the complete network. Since a connection between infected and susceptible hosts defines any potential transmission path for infection, the overall number of connections influences the speed that the epidemic takes a population. Therefore, the total number of connections present in the network regulates the spreading speed of the disease over the network. As the network becomes denser, the system appears more mixed, its behavior approaches a well-mixed population, and spreading kinetics becomes more deterministic. The effects of the network structure are expected to be more evident in sparser networks.

The simulated transmission process is regulated by three model parameters – network structure, transmission probability, and the expected number of pathogens to be transmitted. The disease is successfully transmitted if at least one pathogen migrates from the infected host to a directly-connected susceptible neighbor. Since the contact rate in the discrete SIS model in Equation 10 indicates the number of individuals an infectious host had sufficient contact to transmit the infection, then number of contacts is primarily determined by the number of connections that lead to susceptible hosts d_s (Equation 17). Adding the number of connections to susceptible hosts d_s to Equation 13, if $d_s = 0$, then the effective contact rate β' is 0. On the other hand, if $d_s > 0$, only then will the per-connection transmission probability P_T and the Poisson expected number of pathogens transmitted λ_d matter since they are conditioned on a connection existing and that that connection leads to an infected host.

Equation 17. Effective contact rate of single host in the network

$$\beta' = d_s P_T\left[1 - \mathrm{e}^{-\lambda_d}\right] \tag{18}$$

In the program, successful transmission of pathogens when a connection exists relies on two random events. First, transmission is permitted when the Bernoulli trial draws a 1 given a particular transmission probability. Given a successful draw, another sampling from a Poisson distribution determines the number of pathogens to be transmitted. In order to simplify the process, I set the transmission probability to 1.0 such that the Bernoulli sampling process is skipped and transmission every time the random draw from the Poisson distribution is greater than zero. Since the simulation determines the number of migrants by randomly sampling a Poisson distribution with an expected value λ, then the probability that transmission occurs over a particular connection is simply the probability that at least one pathogen is transmitted over the connection (Equation 13, also Equation 17) given by $1 - \mathrm{e}^{-\lambda_d}$ where λ_d is the expected number of migrants. The probability that at least one migrant transmits increase as λ_d increases. In the simulations, I used values $\lambda_d = 5$ or $\lambda_d = 50$ which makes the contact rate effectively equal to the number of connections to susceptible hosts as the probability given by $1 - \mathrm{e}^{-\lambda_d}$ approaches 1.0.

Another consequence of a complete network is that there is an equal probability for any two hosts to be directly connected to each other such that each host has an equal probability of being connected to another host. If the network topology has an unequal number of connections per host, the contact rate changes with the number of connections. This occurs because in networks with less than a complete set of connections, a transmission path involving a series of hosts cannot exist if there is no connection connecting between an infected and a susceptible host with the series. As a result, hosts with more connections will have a higher contact rates and will spread the infection faster than hosts with fewer connections, given equal transmission probabilities and expected number of migrants.

Previously, I reformulated the contact rate in terms of the number of connections per host. Note that this is only valid under a complete network, or a regular network with randomly changing connections. Moreover, it may not be always valid under a static network where the connections between hosts are constant over time. One case where Equation 17 is not valid occurs when an infected host is surrounded by other infected hosts. The discrete SIS model describe by Equation 10 does not necessarily allow infected individuals to be concurrently reinfected.

In epidemiological compartmental models, hosts located in a compartment move unto the next compartment either due to an inoculation event, stochastic transition based on a transition rate, or transition after spending a given waiting time. If infected individuals are allowed to be re-infected, this creates a kind of “self-loop” relationship within the infected compartment. How this type of relationship is handled depends on rules for transitioning into and out of the infected state. When using transition rate to determine the waiting time within the infected compartment, then re-infection does not change the dynamic if it does not change the transition probability. However, if the system uses a set waiting time, then re-infection is akin to resetting the timer that counts down the time until infection is over. While these simulations use the set duration approach, transmission of pathogens between infected hosts were allowed but was configured such that the waiting time is calculated from the initial inoculation alone.

Site-frequency spectrum of pathogen sequences

Epidemiological studies have shown that network topology and network properties such as network density, degree distribution, and clustering coefficient affect the speed of the outbreak and the proportion of the population affected (Barthélemy, Barrat, Pastor-Satorras, & Vespignani, 2005; Ganesh, Massoulié, & Towsley, 2005; Matt J. Keeling, 2005; Meyers, Newman, Martin, & Schrag, 2003; Moore & Newman, 2000). Here I examine how pathogen molecular evolution can affected by the underlying structure of host networks using the site frequency spectrum.

A usual assumption of the site frequency spectrum is that the sites surveyed to construct the frequency distribution of derived allele frequency classes are independent. By assuming independence among sites, changes in the allele frequency would be solely attributable to state of the site and not dependent on the frequency of the derived allele at another site. In this study, I use the site frequency spectrum method in an unusual way because the sequences I examine are completely linked. Under this scenario, the probability of observing derived alleles are also expected to decrease such that more sites should have singletons or doubletons compared to sites with a high proportion of derived alleles. Since sequences are completely-linked, a high number of sites with high derived allele frequency does not necessarily indicate that the derived allele is advantageous. Sites with high derived allele frequency may have experienced mutations early and have evaded extinction throughout the generations. Thus, a skewed site frequency spectrum compared to the well-mixed null model indicates that the population lacks the expected genetic diversity.

To study the effect of network structure on the pathogen evolution, I considered two different network topologies – regular and scale-free topology and compared the resulting site frequency patterns after 1000 pathogen generations (Figure 12). Like the complete network, the regular network has an equal number of connections per host ($d = 5, 10$), although much less compared to the complete network ($d = 199$). As a result, a 5-regular or 10-regular network will produce a more sparsely-connected network than a complete graph. On the other hand, the scale-free network is characterized by the high variance in the number of connections per host. This occurs because the network is arranged such that a large number of hosts have a few connections while a small number of hosts hold a large number of connections. I also examined if network density affects pathogen evolution. Since high network density imposes less constraint on transmission paths, networks with more overall connections should behave more like unstructured networks. I compared two network densities, dense regular and power-law networks with $D = 0.05$ and sparse regular and power-law networks with half the density ($D = 0.025$).

For the null model, I considered an unstructured model such that the pathogen population completely mixed. In contrast, the network models all use a metapopulation model where the total pathogen population is subdivided into within-host populations. Thus, the unstructured null is simply a Wright-Fisher haploid population. Using this null model, I examined the effects of population structure and network structure on the frequency of new mutations after 1000 generations using the site-frequency spectrum.

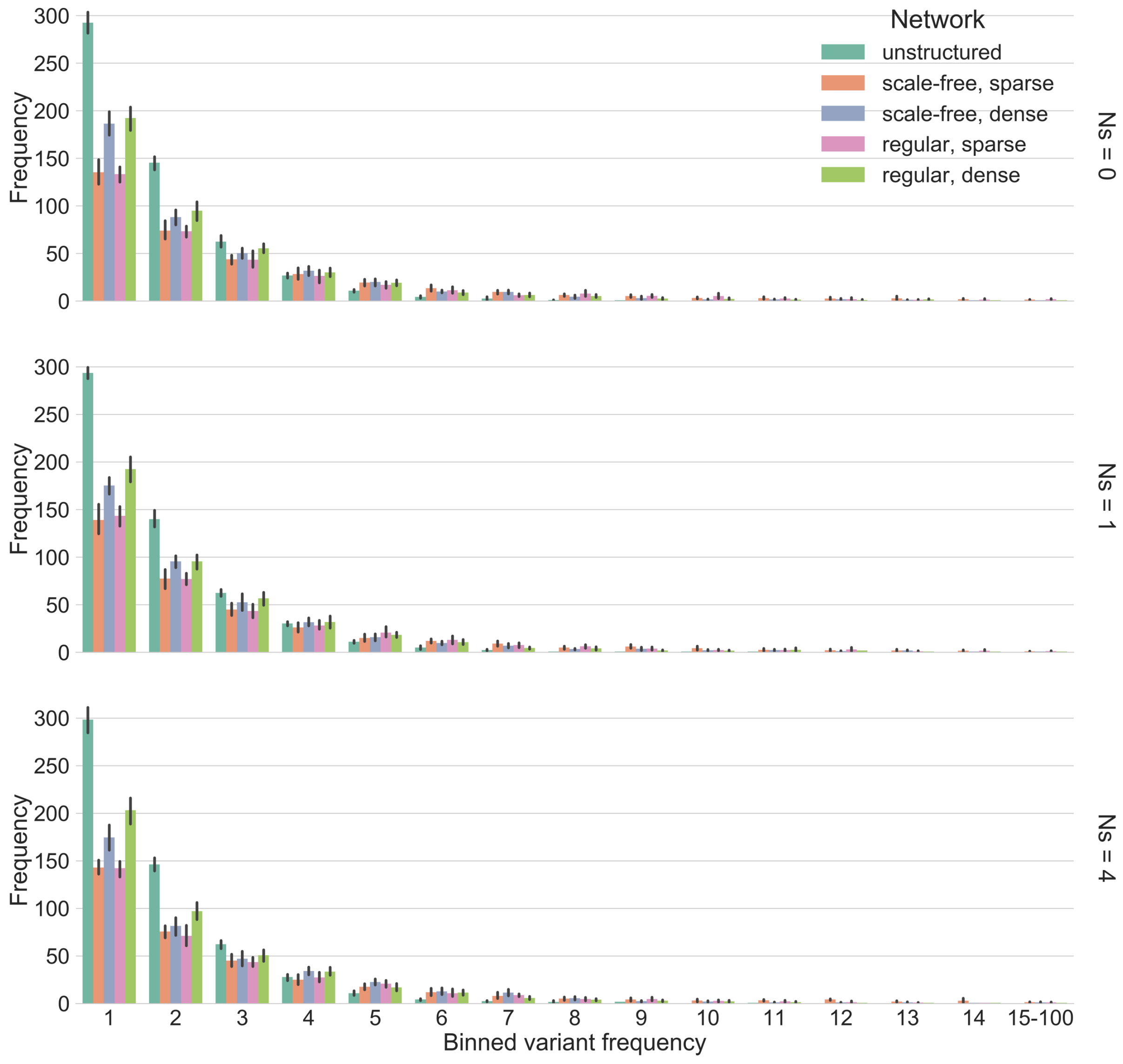


Figure 12. Effect of host network topology on the derived allele frequency spectrum of disease-causing pathogens.

Each site frequency spectrum compares the frequency of site patterns (height) observed when pathogens evolve as an unstructured population (blue green), or when considering the host population network as a sparse (orange) or dense (purple) scale-free network, or as a sparse (pink) or dense (bright green) regular network. The horizontal axis (x-axis) indicates the number frequency of the new mutation from 1 (singleton) to 100 (fixed). Frequency classes are listed individually for the first 14 classes and binned to one class for variants segregating at 15% or greater. The first frequency spectrum describes the frequency pattern when pathogens evolve neutrally (top), while the *second, and third show the differences when fitness differences exist at scaled selection coefficients* $Ns = 1$ *(middle) and* $Ns = 4$ *(bottom).*

A comparison between the unstructured null model and the regular network showed that population subdivision brought about by within-host infection decreases the frequency of rare mutations and elevates the frequency of sites with new variants segregating at higher frequency. Moreover, deviation from the unstructured null model is greater in sparse networks than in dense networks. Sparse networks appear to promote high derived allele frequency sites and reduce genetic diversity regardless of the underlying topology (Figure 12, $Ns = 0$). Note that both the unstructured null model and the regular network showed extremely few sites with high frequency mutants. This is an effect of clonal interference due to the complete linkage of the simulated genetic region. This may also be one reason why differences between network types are less pronounced.

I also examined the interaction between within-host selection and between-host network effects. While higher scaled selection coefficient increased the frequency of observing high derived allele frequency sites in both network and unstructured populations, the rate at which the high frequency patterns increased were most significant in the sparse networks.

Since genetic diversity tends to increase in expanding populations, I tested whether the initial position of the host in the network significantly changes evolutionary dynamics. In a scale-free network, the simulation was initialized in a random location on the network, in a high-degree host whose number of connections belongs to the top 90th percentile, and in a low-degree host whose number of connections belongs to the bottom 10th percentile. Results indicate that initiation bias does not severely change evolutionary outcomes, except for the number of rare mutations which tended to be more frequent in low-degree initializations (Figure 13).

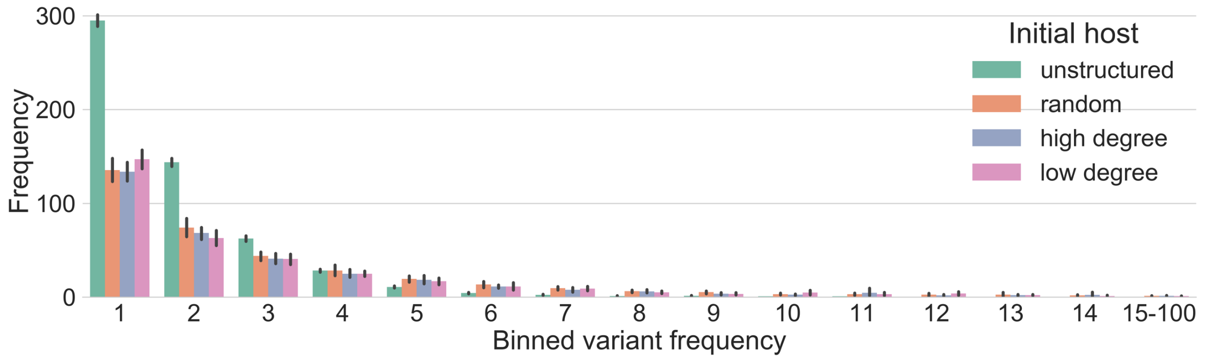


Figure 13. Effect of initially-infected host connectivity on the derived allele frequency spectrum of disease-causing pathogens.

The site frequency spectrum compares the frequency of site patterns observed under neutral evolution ($Ns = 0$) when the epidemic is created from a randomly initiated position in the scale-free network (orange), or when the initial host's number of connections belongs to the top 90th percentile (purple) or the bottom 10th percentile (pink). The frequency spectrum of pathogens as an unstructured population is also shown (blue green) for reference. The horizontal axis (x-axis) indicates the number frequency of the new mutation from 1 (singleton) to 100 (fixed). Frequency classes are listed individually for the first 14 classes and binned to one class for variants segregating at 15% or greater.

Given the static nature of the networks I used, it is possible that infected hosts find themselves surrounded by other infected hosts and become unable to further spread the infection. If the simulation does not allow transmission of pathogens to already infected hosts, this creates evolutionary dead-ends as some pathogen lineages are not transmitted. In this scenario, these neighboring infected hosts behave similar to immune or vaccinated hosts in that they block further transmission of the infection. Although this is not wrong, it is an unexpected side-effect of the no co-infection version of the SIS model.

To study the effect that coinfection has on genetic diversity and pathogen evolution, I simulated the spread of infection using the same model parameters and compared two possible configurations: one where initial infection prevents any secondary inoculation of pathogens while the individual is infected (single infection case), and where secondary inoculation of the pathogen is allowed (coinfection case). For both cases, the infection was allowed to spread in a scale-free network configuration under SIS epidemic dynamics.

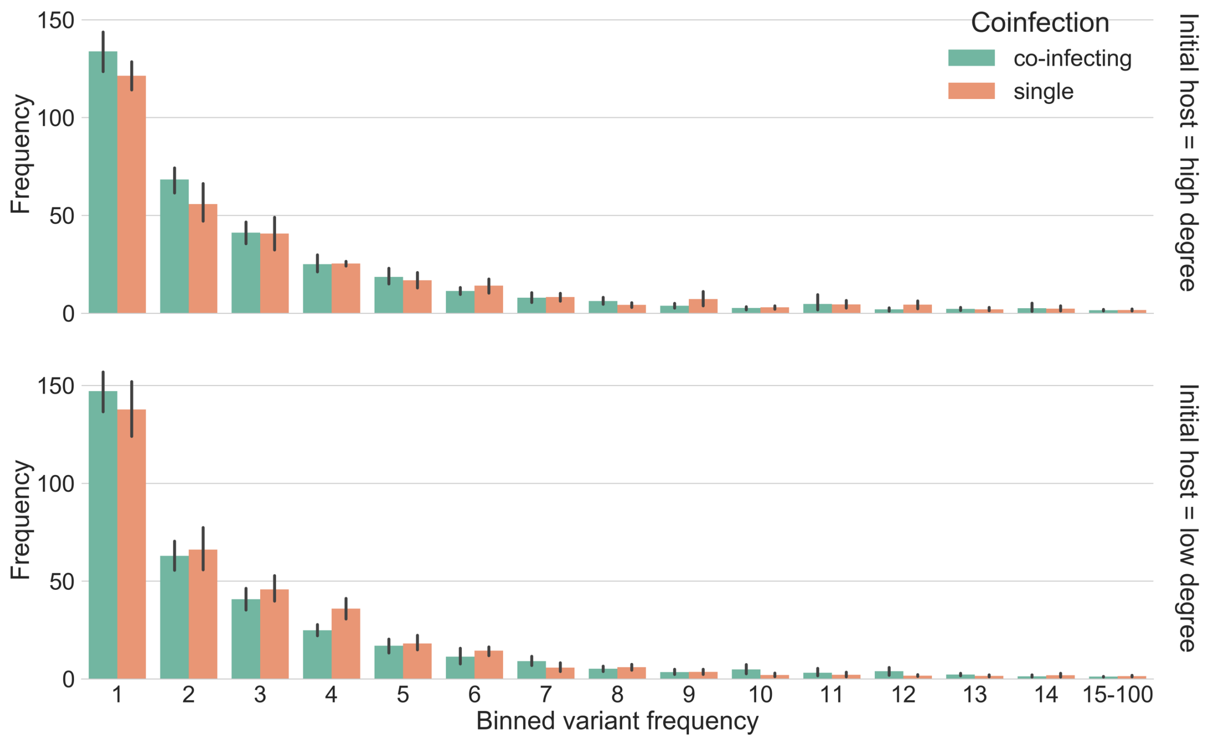


Figure 14. Effect of coinfection on the derived allele frequency spectrum of disease-causing pathogens spreading over scale-free host network.

Each site frequency spectrum compares the frequency of site patterns observed when infected hosts are allowed to transmit pathogens to other infected neighbors (blue green), or when the infected hosts are restricted to transmitting pathogen only to susceptible neighbors (orange). The top frequency spectrum shows the differences when the epidemic is initiated from a host whose number of connections belongs to the top 90th percentile (high degree), while the bottom frequency spectrum shows the differences when the epidemic is initiated from a host whose number of connections belongs to the bottom 10th percentile (low degree). The horizontal axis (x-axis) indicates the number frequency of the new mutation from 1 (singleton) to 100 (fixed). Frequency classes are listed individually for the first 14 classes and binned to one class for variants segregating at 15% or greater.

Allowing coinfection to occur significantly changes the distribution of genotypes compared to single infections only (Figure 14). Coinfection increases the number of sites with low frequency variants compared to the single infection system. Moreover, the presence of coinfections increases genetic diversity regardless of the initial host's connectivity to the population. This indicates that the single infection scenario increases the effect of network structure in pathogen evolution.

Fixation probability in different network configurations

The skewed patterns of the site frequency spectrum for networks compared to the unstructured model suggests that some characteristic of the network may be influencing the fixation of mutants. To test this hypothesis, I examined the fixation probability of a mutant pathogen spreading over different types of network from a single host at an initial within-host frequency $p = 0.5$ and within-host pathogen population size $N_d = 500$ (Figure 15). Likewise, I compared the network scenario to the fixation of a mutant in an unstructured population whose initial frequency was also set to $p = 0.5$ and initial pathogen population size of $N_d = 500$ (Figure 15, light gray).

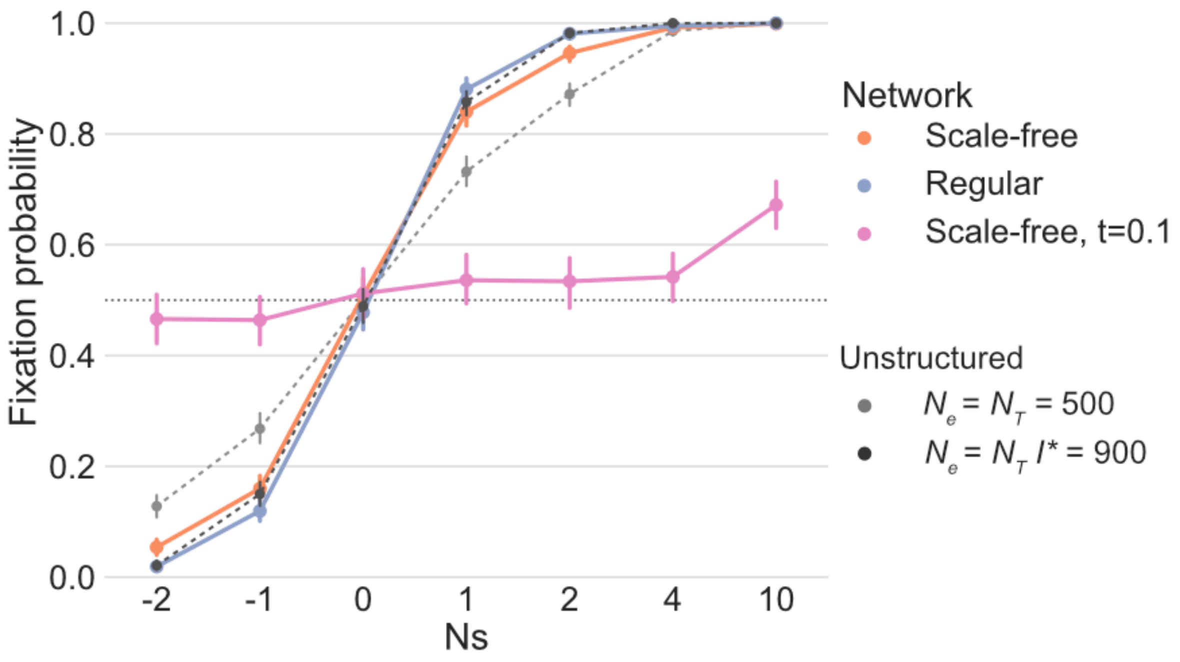


Figure 15. Effect of network topology on the fixation probability of a mutant starting from an initial frequency p=0.5.

Fixation probabilities of the P-genotype pathogen spreading over a scale-free (orange) and regular (purple) host network over a range of scaled selection coefficients (x-axis) are shown above. Fixation probabilities of in a scale-free host network when transmission potential is low (pink) is also included. The fixation probabilities of an unstructured population of pathogens for $-2 \leq Ns \leq 10$ *when* N *is equal to the within-host population size (light gray) and equal to the product of the equilibrium number of infected hosts and the number of pathogens transmitted (dark gray) are included for reference.*

Network models of pathogen evolution showed elevated fixation probabilities for positive scaled selection coefficients ($Ns = N_d s; Ns > 0$) and lower probabilities of deleterious mutants

($Ns < 0$) reaching fixation. However, under neutral evolution ($Ns = 0$), the average fixation probability remained unchanged contrary to the patterns observed in the site frequency spectrum analysis. When selection is present in the system, regular networks showed slightly higher probability of fixation compared to scale-free networks, and both were significantly higher compared to the unstructured null model (Figure 15).

In this simulation, pathogen sampling per generation is not constant. When pathogens are within hosts, they maintain a constant within-host population size N_d. However, when pathogens are transmitted to susceptible hosts, the sampling size is reduced to mean of 5 pathogens per transmission. Thus, this reduction may also be affecting the fixation probability in the network cases. To address this possibility, I also simulated an unstructured population where the population size is equal to the product of the transmission size $k = 5$ and the number of infected hosts $n = 180$ such that $k \times n = N = 900$. Results show that the network cases when transmission is high varies around this $k \times n$ unstructured population. This indicates that network structure elevates the fixation probability when degree is regular but reduces the fixation probability when the degree is highly heterogeneous.

As an extreme case, I also included an approximation to model the low contact rate as a result of the sparsity of connections in sparse network (Figure 15, pink line). Sparse networks cannot be directly simulated for the set of parameters because the smaller host population size prevents the network from being too sparse such that some hosts are unconnected from the rest of the population. In order to approximate the effects of sparsity, I reduced the per-connection transmission probability such that the effective contact rate is equivalent to the contact rate in sparse scale-free network in the previous analysis. Fixation of this extreme case reinforces the observation that low contact rate makes it less likely that a pathogen lineage survives, let alone spread throughout the host population network and become fixed.

To explain the discrepancy between the skewed site frequency spectrum patterns under neutral evolution and the consistent fixation probability of network and unstructured models when $Ns = 0$, I considered whether evolution over networks changes the time until mutants become fixed.

To find the time to first fixation conditioned on fixation, I examined all realizations where the mutant successfully reached fixation ($p = 1.0$) and recorded the number of pathogen generations it took until reaching fixation. I considered the time to fixation for the unstructured null model and both regular and scale-free network models. I also included the approximate sparse network model to show how an extremely limited supply of contacts affects fixation time.

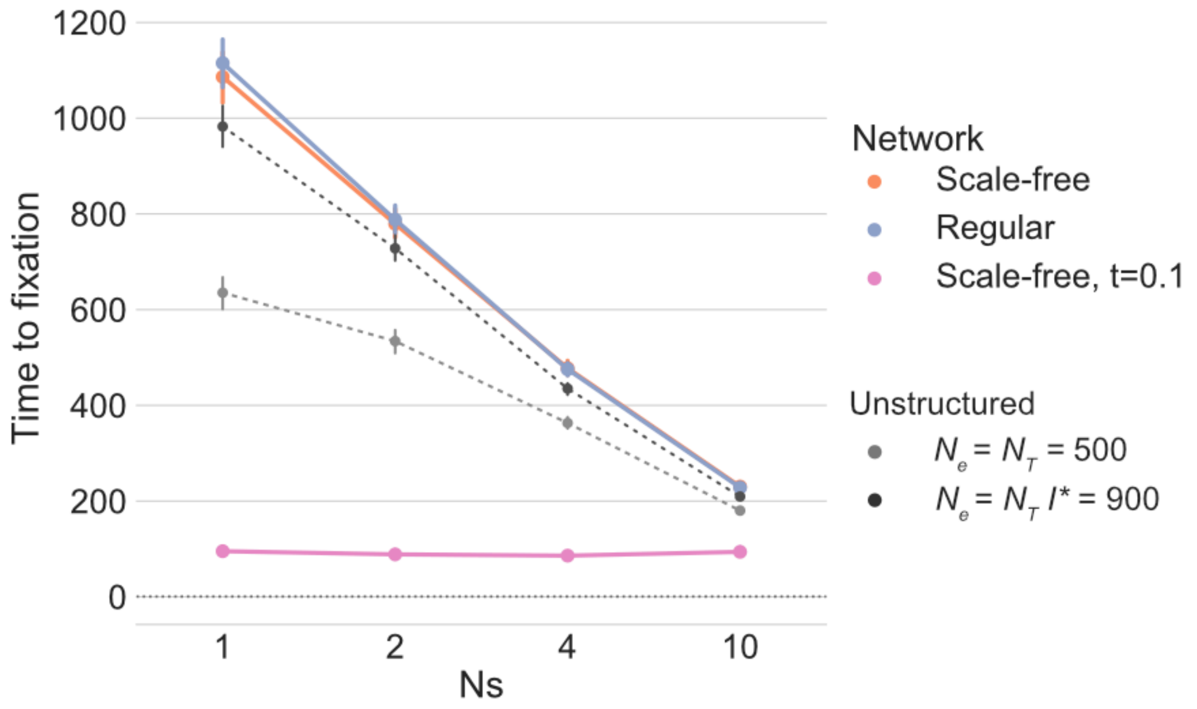


Figure 16. Effect of network topology on the time it takes for a mutation from an initial frequency of p=0.5 to reach fixation, conditioned on fixation.

The average time for a P-genotype pathogen to reach fixation given it reaches fixation is shown for scale-free (orange) and regular (purple) host networks. The average time to reach fixation conditioned on fixation for pathogens spreading over a scale-free host network when transmission potential is low (pink) is also included. The average time to reach fixation conditioned on fixation in an unstructured population of pathogens for $-2 \leq Ns \leq 10$ when N is equal to the within-host population size (light gray) and equal to the product of the equilibrium number of infected hosts and the number of pathogens transmitted (dark gray) are included for reference.

Time to fixation analysis revealed that network models took more than 1.5 times longer to reach fixation compared to the unstructured population. This result seems to be in direct disagreement with the frequency spectrum patterns. If mutants spreading over a population networks took a longer time to fix than an unstructured population, then networks should have an excess number of singletons even greater than the unstructured model. The network model took a longer time to fixation mutants because of the semi-isolated structure imposed by the infection of pathogens within hosts in the population.

This semi-isolated structure could be one possible explanation that resolves the seemingly conflicting results between the frequency spectrum and the fixation analysis. Hosts encapsulate pathogens during infection and pathogen populations only mix when coinfection occurs, creating a highly-structured population where lineages that arose in separate host infections rarely mix together. At the same time, within-host isolation allows pathogen mutants to reach intermediate levels of frequency as observed in the site frequency spectrum patterns.

DISCUSSION

The network model in pathogen evolution isolates pathogens populations within hosts which creates pathogen lineages that rarely interact. This model structures the overall pathogen population into discrete subpopulations of pathogens residing within infected hosts. As a result, the expansion of pathogen lineages across different hosts are more likely to be determined by the transmission potential of hosts rather than an intrahost fitness advantage of pathogen. A comparison between dense and sparse networks showed that structure of the network had a larger impact on the frequency of alleles compared to intrahost selection.

The route of transmission of a disease is a key determinant whether network models are necessary, or an unstructured population can be assumed. For example, Influenza can be transmitted directly from expectorated aerosol particles or through infected surfaces, and requires close contact in order to transmit the infection (Kawashima et al., 2016). In cases of close contact, the probability that any two random hosts in a population will come into contact such that the disease transmits is low and the probability of contact between hosts is not even distributed in the population. Geographically, it is more likely for hosts that live, work or go to school in the same area to come into contact. Social behaviors also affect the probability of host contact. Moreover, social behaviors such as meeting friends, going to big events such as concerts or sporting events will tend to segregate individuals by age and social groups. These factors promote heterogeneity in the population and in these instances, the effect of network structure will become more pronounced.

In contrast, carrier-borne diseases such as cholera can be contracted without ever meeting a sick individual because transmission is mediated by a vector, in this case, water. Because of the way these diseases spread, pathogens that cause these infections are not always siloed within individual hosts and have more opportunity to comingle in the environment. Results indicated that network effects occur because pathogen lineages almost never interact except in rare instances of mixed infection within a single host. When pathogens were allowed to freely mix within hosts, as in the case of the coinfection experiment, the frequency distribution of alleles more closely resembles the unstructured null model. Although it is tempting to consider that a complete network should approximate the unstructured population, this is not entirely correct. It is correct to assume that a completely-connected network more closely resembles the unstructured population. However, the network model still encapsulates pathogens within hosts and restricts free movement between hosts by imposing a transmission bottleneck. So while the capacity to transmit to every other hosts is present in the complete network, there is still little free movement across hosts.

Allowing coinfection in the system creates more paths for transmission available to pathogens compared to when coinfection is not allowed. Similar to the effect of higher network density on the frequency spectrum, the coinfection makes the system behave closer to the unstructured model while preventing coinfection increases the effects of isolation and population structure. Under this coinfection model, an infected host can transmit pathogens to an infected neighbor, but the secondary infection does not reset the infection time started by initial infection. This mean that the duration of infection is set by the initial infection and is not affected by subsequent inoculations while infected. Physiologically, this scenario assumes that pathogen strains are not immunological distinct such that response mounted for the first infection also helps defeat secondary exposure and infection.

New technology has facilitated rapid and abundant sequencing of pathogen genomes. Pathogen sequencing has been used to study the spread of virulence genes in E. coli, plot the geographical distribution of Dengue viruses using gene and genomic data, and track within-host evolution of Hepatitis C virus during treatment. These studies focus either on the evolution of pathogens across large landscapes for an extended period of time or examined evolution within-hosts.

CHAPTER 4
PERIODIC INFECTION AND TRANSMISSION PARAMETERS AFFECTS THE FIXATION OF MUTATIONS

INTRODUCTION

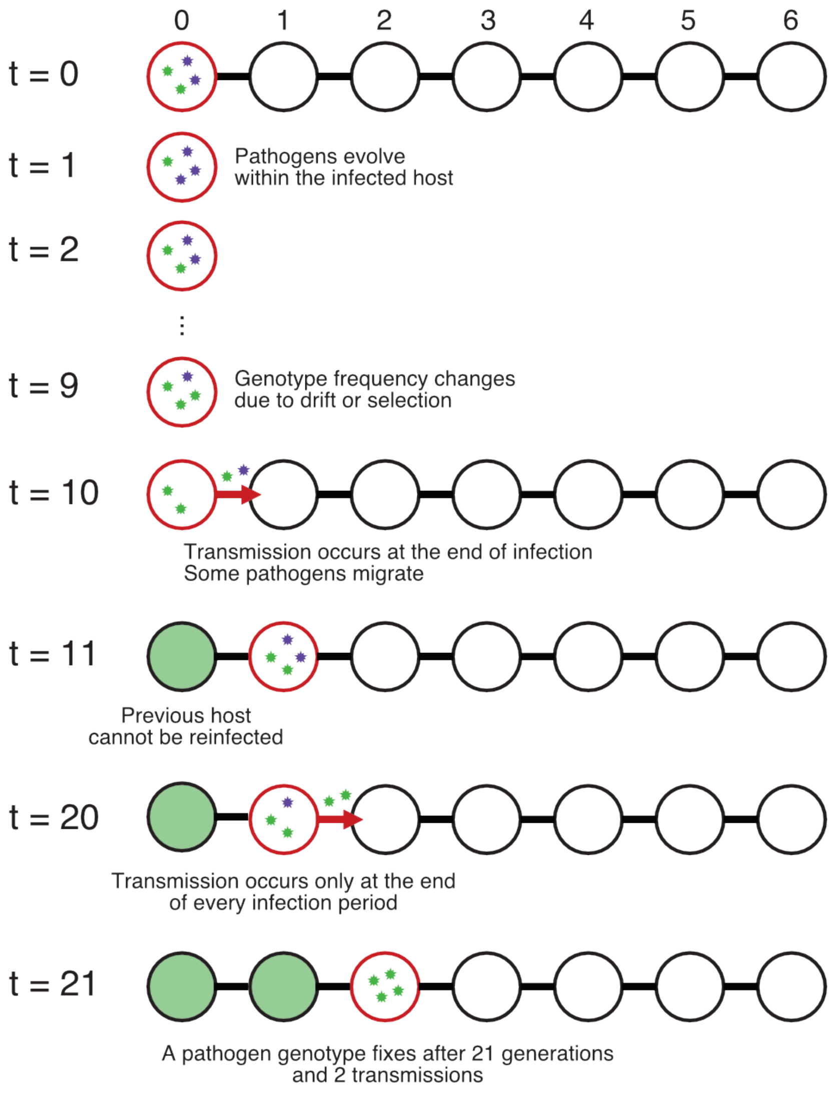
In viral pathogens, reports of positive selection have been based on a collection of sequences sampled from different hosts. Given that selection, in the traditional sense, occurs within hosts, then sites under positive selection must also be evident in within-host samples (Kennedy & Dwyer, 2018; Luciani & Alizon, 2009; McCrone et al., 2018; Park et al., 2013). If present, natural selection should rapidly elevate the frequency of new beneficial mutations during an infection such that particular genotypes become overrepresented. However, deep-sequencing of intrahost viral pathogen populations showed that intrahost single nucleotide polymorphisms are rarely observed (McCrone et al., 2018; Murcia et al., 2010; Poon et al., 2016), indicating that viruses are under strong purifying selection (Holmes, 2003). Moreover, experimental evidence of positive selection occurring within hosts remains inconsistent. When positive selection is observed in within-host populations, it has been limited to cases of drug resistance from ongoing treatment (Ghedin et al., 2011) or under exceptional conditions (Xue et al., 2017).

If within-host natural selection is a rare event, can it sufficiently explain the signature of positive selection observed at the host population level? On the other hand, it is possible that the detected signal is not due to natural selection but is an artefact produced by the transmission process. This study attempts to reconcile these conflicting results by connecting the within-host process of selection and between-host process of transmission and measuring their influence on fixation probability.

METHODS

Transmission chain model

The transmission chain model is a specialized scenario based on susceptible-infected-removed (SIR) epidemic model spreading on a linear network of hosts. The goal of this model is to simplify the transmission process from an expanding random tree to a linear network by limiting the potential paths of spreading and decreasing the randomness in the process (Figure 17).



0
1
2
3
4
5
6
t = 0
t = 1
Pathogens evolve within the infected host
t = 2
t = 9
Genotype frequency changes due to drift or selection
t = 10
Transmission occurs at the end of infection Some pathogens migrate
t = 11
Previous host cannot be reinfected
t = 20
Transmission occurs only at the end of every infection period
t = 21
A pathogen genotype fixes after 21 generations and 2 transmissions

Figure 17. Diagram depicting the dynamics of the transmission chain model.

The diagram illustrates the infection of the initial leftmost host at t=0 and the changes in genotype frequencies within the infected host during the infection. During this time, no mutation occurs and the number of pathogens within the host remains unchanged at N_d (constant population size). At t=10, the infection is transmitted to the neighboring host to the right of the initially infected host. This *transmission event randomly selects pathogens to be transmitted and inoculates the pathogens into the susceptible uninfected host. At t=11, the initially infected host recovers, and the newly infected host becomes instantaneously infected with the pathogen with a population size N_d. The infection and transmission processes are repeated until one of the genotypes is fixed.*

In the transmission chain model, a single initially-infected host is located at one end of the linear network while the rest start from a susceptible state. Susceptible individuals become infected if they have been inoculated by at least one disease-causing pathogen. When at least one pathogen is present within the host, infection progresses for given time interval specified by the number of pathogen generation since transmission, during which the pathogen population size N_d stays constant. During infection, pathogens replicate faithfully without mutation and may be under selection if fitness differences exist among the pathogen genotypes in the within-host pathogen population. To determine the next generation of pathogens, the fitness of each pathogen is computed based on the pathogen's genotype and a given multiplicative fitness matrix. During infection, disease-causing pathogens can also be transmitted from an infected host to a susceptible neighbor. When transmission occurs, the infected source adopts the removed state and the inoculated host becomes infected in the next generation. Given the linear topology of the network, this means that pathogens transmit only to the next adjacent susceptible host. Pathogens are sampled from the currently infected host randomly such that the probability of sampling a particular genotype is frequency-dependent. The timing of the transmission event can be decided stochastically by sampling from an exponential distribution or occur after a specific time based on the model specification. After the infection time interval has elapsed, the infection present in the infected individuals is removed by removing all pathogens present within the host. As a rule of the SIR model, previously infected hosts become immune from the infection and remain vaccinated until the end of the simulation.

Transmission tree model

The transmission tree model focuses on the effects tree shape on the evolution of pathogens. This is a more complicated version of the transmission chain model that models hosts embedded in a tree structure instead of a linear network. In this model, spreading of the disease occurs deterministically and is conditioned on the provided tree topology. Compared to simulating over the entire population network, the transmission tree model provides a way to control the shape and timing of transmission in order to provide model consistency over independent realizations (Figure 18).

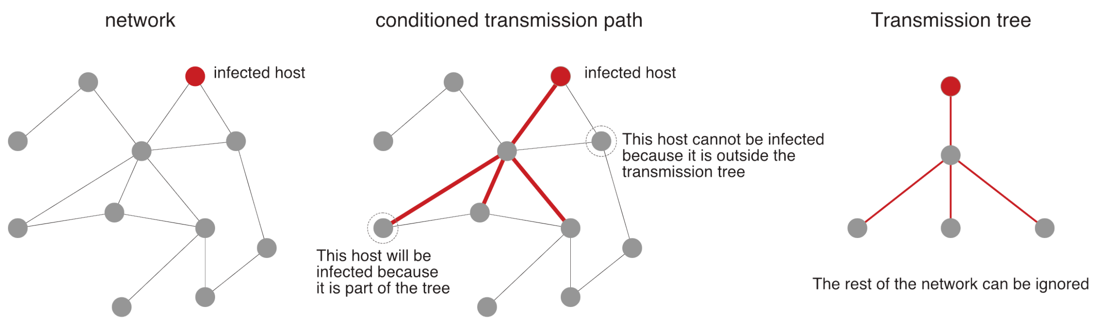

network
infected host
conditioned transmission path
infected host
This host cannot be infected because it is outside the transmission tree
This host will be infected because it is part of the tree
Transmission tree
The rest of the network can be ignored

Figure 18. The transmission tree is the set of paths in the network where transmission is conditioned to occur.

Given a set of connected hosts (gray dots connected by edges), an infected host (red dot) may transmit the disease to others in the population (left). The transmission tree model takes from percolation theory and conditions the network to include pre-selected the paths that the disease will transmit on (center). Parts of the host population network that does not experience the disease are ignored, leaving only the transmission tree (right).

One way to imagine the relationship between a transmission tree and the network is to think about the set of paths pathogens will take. Normally, the direction of the spreading is determined randomly given the set of connection in an infected host. However, another way to think of it is to randomly pick the paths the infection will take all at once. Any host within in this path is guaranteed to be infected. This is equivalent to having a network structure present and pre-selecting the path of transmission all at once instead of one at a time (Figure 18). Because parts of the network that do not experience the infection do not contribute to the dynamics of the epidemic, these hosts can be ignored.

Fixation probability of a mutation over successive transmissions

To test the effects of transmission size and the duration of an infection on fixation, I repeatedly simulated successive transmissions under the transmission chain model and measured the number of fixations events that occurred. I considered a system where pathogens have two possible genotypes, U and P. Initially, the frequency of U and P in the initially-infected host was set to the equilibrium frequency $p = 0.5$ and pathogen population size within the host was $N_d = Ns = 500$. A total of four different simulations were created based on two transmission size of $k = 5, 50$ representing 1% and 10% population size bottlenecks, and two duration lengths of infection $t_I = 10, 20$ pathogen generations. The chosen lengths of time for infection were chosen given that the acute viral infections in human typically last between one to two weeks (Fields et al., 2013). To convert this into pathogen generations, I considered one lytic cycle, which is the time it takes for an infected cell to lyse and disperse new viral particles, as one pathogen generation. In many human viral infections, the lytic cycle *in vivo* and *in vitro* approximately takes one day to complete (Fields et al., 2013). Therefore, simulation time is equivalent to one lytic cycle when considering viral pathogens, and approximately equal to one 24-hour period in real time. Transmission from an infected host to its susceptible neighbor was limited to occurs only at the end of the infection. Simulations were repeatedly performed and the number of fixation events over 1000 independent realized were used to get the probability of fixation.

The empirical fixation probability was also compared to the theoretical value that assumes a homogenously-mixing haploid population of pathogens that does not experience population bottlenecks using the equation derived by Kimura and Ohta (1969) adapted for haploid populations (Equation 16).

Proportion of successful fixations of a mutation in an expanding transmission tree

To test the effects of transmission shape on the fixation of mutations, I repeatedly simulated pathogen transmissions under the transmission tree model and measured the number of fixations events that occurred after 1000 pathogen generations.

I considered three tree topologies that depict the different transmission trees that can be formed when diseases spread over a host population network (Figure 19). The regular tree (Figure 19a) represents equal rates of spreading of the infection for every infected host, which is equivalent to a network with a singular number of connections per host. Thus, this tree topology simulates a scenario where the contact rate between infected and susceptible hosts is uniform across hosts. The second tree topology represents the scenario when spreading is not completely regular such that the contact rate for each host depends on the number of connections (Figure 19b). The third tree represents the rapid change in the prevalence of the disease caused by a superspreading event (Figure 19c). The superspreader tree is extreme case of heterogeneity in contact rates caused by differences in the number of host connections.

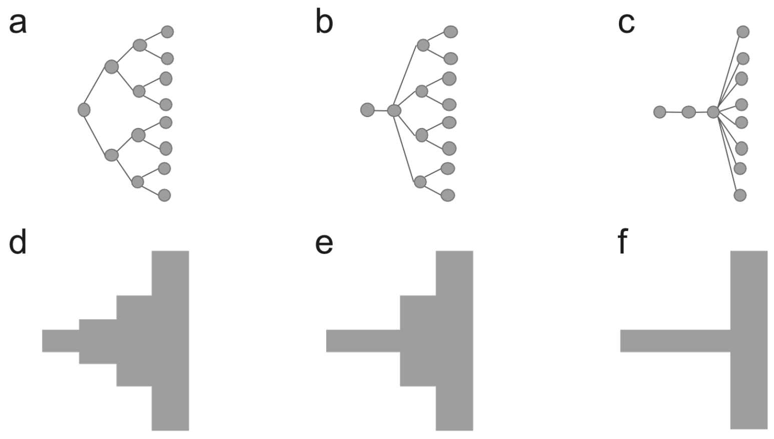


Figure 19. Transmission trees used in the simulation study.

(a-c) Diagram of transmission tree topologies used. Nodes represent host individuals while edges represent the transmission from on one host to another. Three transmission tree topologies were considered: (a) regular, (b) heterogeneous, and (c) superspreader. (d-f) Box heights illustrate the relative population sizes for each level of the corresponding transmission tree.

In this system, I initialized the simulation with a single infected host with a pathogen population size $N_d = 500$, and whose pathogen genotype frequencies were set to the equilibrium frequency $p = 0.5$. Similar to the simulations over linear networks, transmission from an infected host to its susceptible neighbor was limited to occurs only at the end of the infection. The duration of infection was set to 250 pathogen generations and transmitted five randomly-sampled pathogens (10% population bottleneck) to each susceptible neighbor when a transmission event occurs. When the host is infected, pathogens replicate at the rate of their replicative fitness based on their genotype while keeping a constant within-host pathogen population size. No new mutations are introduced during the simulation such that present variation is a result of initial standing variation and the effect of selection.

To isolate the influence of the tree structure on pathogen evolution, a null model for each tree topology was constructed using an unstructured population that mimics the population size changes of the system over time (Figure 19 d, e, and f). To simulate the unstructured population, pathogens were modeled as a completely-mixed population with changing population sizes over time.

All cases were repeatedly performed for 1000 pathogen generations and the number of fixation events over 1000 independent realizations were used to get the probability of fixation. The 95% confidence interval for each simulation experiment was constructed by bootstrapping the 1000 endpoints 1000 times and tallying fixation for each resampled set.

RESULTS

Transmission events and the fixation probability of a mutation

Transmission events between infected and susceptible individuals can be imagined as a migration of pathogens emigrating from the infected host and founding a new population in the susceptible host (Figure 17, $t = 11$). Although the typical number of pathogen particles involved in transmission is unclear, studies in viruses have shown that even a single viral particle is capable of successfully mounting an infection in the host and thereby founding a new population of viruses in that host (Zwart et al., 2009).

Population bottlenecks have been shown to reduce genetic diversity and increase genetic drift in the population (Bergstrom, McElhany, & Real, 1999; Hongye Li & Roossinck, 2004; Tajima, 1989). The reduction in genetic diversity after a bottleneck depends on the severity of the population size reduction and the length of time the population experiences a reduced population size (Otto & Whitlock, 1997). In the case of pathogen evolution, population bottlenecks do not only occur once or twice in the pathogen's history, but instead is a recurring event that happens every time the pathogen is transmitted from an infected to a susceptible host.

Here, I measured the effect of recurrent population bottlenecks on the fixation probability of mutants in the pathogen population (Figure 20). When pathogens are neutrally evolving, there was no change in the fixation probability as expected. At scaled selection coefficients greater than zero ($Ns > 0; N = 500$), recurrent bottlenecks significantly decrease fixation probability when selection is weak but recovers when selection is strong ($Ns = 10$). On the other hand, at scaled selection coefficients less than zero, recurrent bottlenecks significantly increase the probability that deleterious mutations reach fixation compared to a population that does not experience recurrent bottlenecks.

The number and proportion of migrants in relation to the within-host population size determines their effect on fixation of mutants. At a within-host population size $N_d = 500$, transmitting $k = 5$ or $k = 50$ pathogens at every transmission event represents a 1% and 10%

population bottleneck, respectively. With the same length of infection (Figure 20, same color across charts), results show that a smaller transmission sizes more severely impact the probability that mutants reach fixation compared to when there is a higher number of migrants.

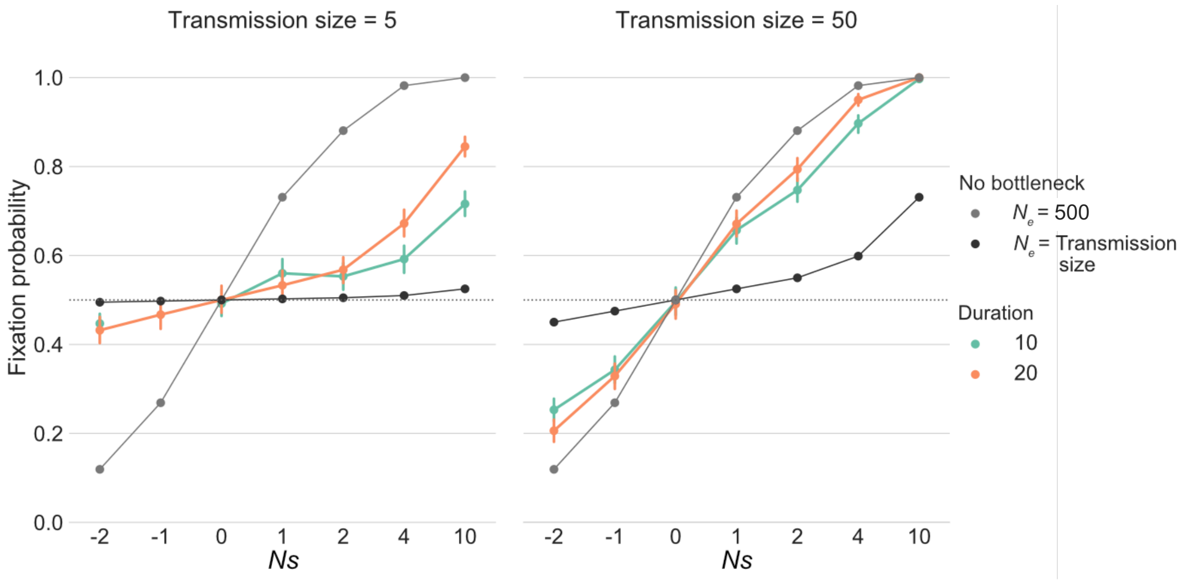


Figure 20. Fixation probability under different transmission and infection parameters.

Fixation probability at different scaled selection coefficients when 5 pathogens – equivalent to 1% of the within-host population – (left) or 50 pathogens – equivalent to 10% of the within-host population – are transmitted between infections. Green and orange points indicate the fixation probability when the infection lasts for 10 or 20 pathogen generations respectively. Light gray points indicate the fixation probability when the population does not incur bottlenecks because the transmission size is equal to within-host population size, while dark gray points show the fixation probability when the within-host population size is reduced to match the indicated transmission size.

I also examined whether the length of time an infection persists in the host affects the fixation probability of mutations. Disease infections have different mean durations of infection depending on the type of infection, the condition of the individual and other physiological and immunological factors (Antia, Ganusov, & Ahmed, 2005; Fields et al., 2013; Murillo et al., 2013). Since pathogen evolution occurs during infections, the length of time an infection occurs regulates the number of generations pathogens evolve and influences the eventual frequency of mutants within the host.

In this experiment, I considered two different time intervals based on the duration of acute infections in humans and represents the time interval between population bottlenecks (Figure 20). At $\Delta t_I = 10$ pathogen generations, the fixation probability is significantly lower compared to when

$\Delta t_I = 20$, which is twice the length of time. Moreover, when the number of pathogens transmitted is controlled (Figure 20, same panel), infections of longer duration tend to have a higher fixation probability than infections of shorter duration. A comparison between the two intervals show that the change in fixation probability depends on the scaled selection coefficient governing selection between the two variants. When $|Ns| > 0$ and as $|Ns|$ increases, the effect of infection duration becomes more pronounced. At larger Ns, variants receive a greater fitness benefit which results in a higher probability of the genotype continuing into the next generation. As a result, pathogen genotype with higher replicative fitness are also more likely to be sampled during transmission events. This suggests that both the transmission size and the infection length are important side effects of transmission events that affect pathogen evolution. Pathogen transmission is a necessary process that creates recurrent phases of inoculation, infection, and transmission resulting in unavoidable periodic extreme changes in population size.

Given these results, I compared simulation data to analytical calculations based on Equation 16 in order to compare to populations with a constant population size that do not experience any bottleneck. I considered two cases: transmission size is equal to the intrahost population size such that $N = N_d = 500$ (Figure 20, light gray), and population size is reduced to the transmission size such that $N = k$ where $k = 5$ or $k = 50$ depending on the transmission size used (Figure 20, dark gray). In relation to the simulation results, these two cases appear to form an upper and lower bound for fixation probability. The shorter infection duration approaches to the $N = k$ case whereas the longer duration is closer to the $N = N_d$ case and is consistent for both transmission sizes tested.

To explain these results, let us consider the two variables independently. First, when infection duration is longer, the pathogen lineage will undergo fewer transmissions and therefore fewer population bottlenecks. Longer infection times also increases the chance that a genotype will fix before the first bottleneck. In the case of transmission size, let us assume the is an equal probability of picking any pathogen particle for transmission. This means that genotype does not bias the probability of selecting a pathogen for transmission and can only affect its frequency in the population. Under this assumption, the larger the number of pathogens transmitted, the more likely the sampled set will be representative of the infection population. On the other hand, if the number of transmitted pathogens is small, then the random nature of selecting migrants will make it less likely to preserve the existing distribution of genotypes. The extreme case of this is when only one pathogen is transmitted.

The periodic increase and decrease of pathogen population sizes over several infections and transmissions of different hosts is conceptually similar to a cyclical fluctuation in population size in a single environment. Previously, Otto and Whitlock (1997) have shown that the harmonic mean of the population size over time can be as an approximate effective population size. Using this method, I compared the fixation probabilities from simulated data to the theoretical fixation probability using the harmonic mean as the effective population size in Equation 16.

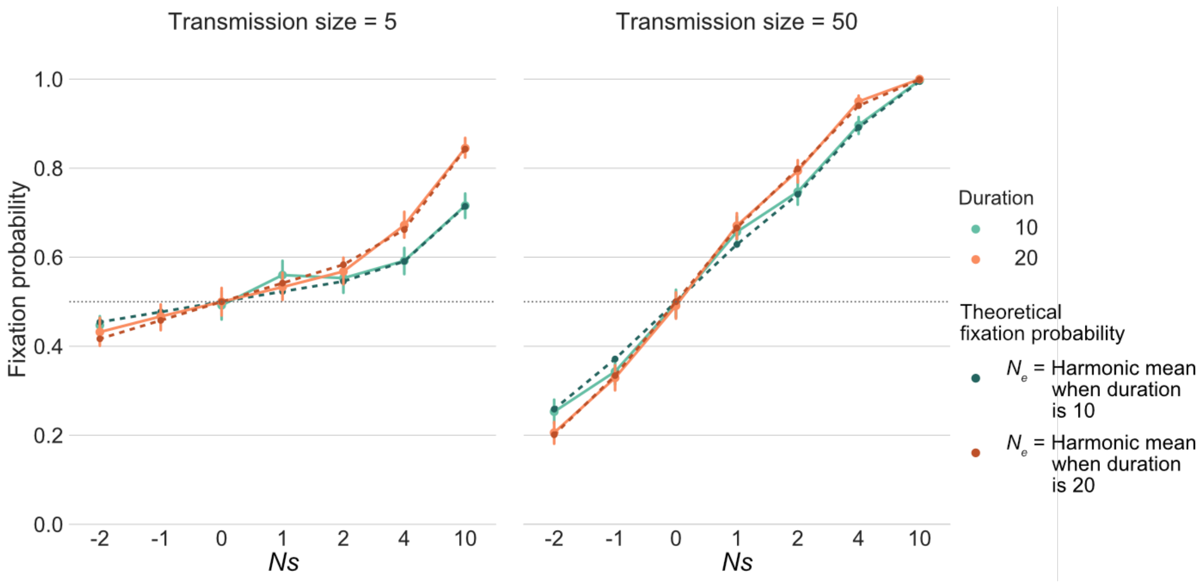


Figure 21. Fixation probability of the infection-transmission process depends on the harmonic mean of the sample sizes over time.

Fixation probability at different scaled selection coefficients when 5 pathogens – equivalent to 1% of the within-host population – (left) or 50 pathogens – equivalent to 10% of the within-host population – are transmitted between infections. Green and orange points indicate the fixation probability when the infection lasts for 10 or 20 pathogen generations respectively. Dark green and dark orange points indicate the fixation probability using the harmonic mean of within-host population size and respective transmission sizes in an infinite amount of time.

Interestingly, the theoretical fixation probabilities predict the simulated data quite closely (Figure 21). In the simulations, the duration of the infection is a Poisson random variable with an expected value of either 10 or 20 pathogen generations. At the end of infection, the pathogen population is transmitted to the next susceptible host. Thus, the average waiting time between inoculation of the pathogen into the host and transmission to the next host equal to the mean duration of infection and is also a Poisson variable. For an infection duration of $\Delta \mathrm{t_I} = 10$, pathogens in the current generation are sampled 9 times at a sample size equal to the within-host population size N_d and sampled once at a sample size equal to the transmission size. Given the periodic nature of this pattern, at infinite time, the pathogen population will have a sample size equal to the within-host population size N_d at $\Delta \mathrm{t_I} - 1$ out of $\Delta \mathrm{t_I}$ times, and a sample size equal to the transmission size one out of $\Delta \mathrm{t_I}$ times. From here the harmonic mean can be calculated as:

Equation 18. Harmonic mean of the pathogen population size at infinite time.

$$\widetilde{N} = \left[\frac{{N_T}^{-1} + \sum_{t=1}^{\Delta t_I - 1} {N_d}^{-1}}{\Delta t_I}\right]^{-1} \tag{19}$$

This harmonic mean approximation works as long as the number of generations where the population size is either increasing or decreases is significantly less than the time scale where selection occurs (Otto & Whitlock, 1997). Given that the simulations take only one generation for every infection cycle to transmit and found a new population, the fluctuations are instantaneous, and is well-suited for this type of approximation.

Recurrent transmissions and the amount of time to reach fixation

I analyzed the effect of recurrent bottlenecks caused by repeated transmissions on the time it takes for a mutation to reach fixation conditioned on fixation (Figure 22). This quantity describes the length of time a site remains polymorphic from an initial frequency p until the allele of interest reaches a frequency $p = 1.0$. Note that the average time to fixation conditioned on fixation describes the average arrival time to $p = 1.0$ for all successfully fixation events only. Because of this, simulations under neutral evolution starting from an initial frequency of $p = 0.5$ will produce only around 50% successful fixation for any given number of trials, while strong selection cases will produce close to 100%. This can be observed in the elevated variances for cases of neutral evolution and weak selection. Variance decreases as the number of successful fixations increases, which increase proportional to the value of the scaled selection coefficient.

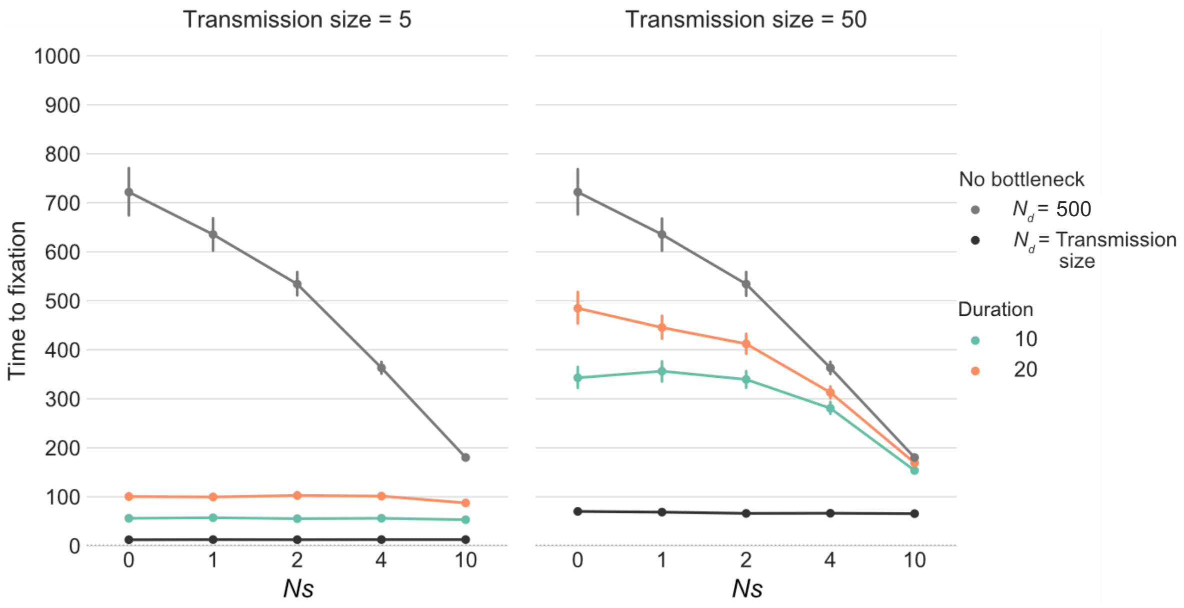


Figure 22. Time to fixation conditioned on fixation under different transmission and infection parameters.

Time to fixation given fixation for different scaled selection coefficients when 5 pathogens – equivalent to 1% of the within-host population – (left) or 50 pathogens – equivalent to 10% of the within-host population – are transmitted between infections. Green and orange points indicate the length of time until fixation when the infection lasts for 10 or 20 pathogen generations respectively. Light gray points indicate the length of time when the population does not incur bottlenecks because the transmission size is equal to within-host population size, while dark gray points show the length of time when the within-host population size is reduced to match the indicated transmission size.

When transmission size is small ($k = 5$), the average time to fixation from an initial frequency $p = 0.5$ is not significantly affected by the selection (Figure 22). Results show for a transmission size $k = 5$ and $-2 \leq Ns < 10$, fixation appears to be facilitated by 5th transmission event given that fixations predominantly occur at the 50th and 100th generation for $t_I = 10$ and $t_I = 20$ respectively. However, when the transmission size is large, the average time to fixation from $p = 0.5$ becomes sensitive to within-host selection and is inversely proportional to the strength of selection. In both cases, recurrent transmission hastens the fixation of mutations due to periodic population bottlenecks.

In the two scenarios where no population bottleneck takes place, the times to fixation over the range of scaled selection coefficients appear to form upper and lower limits for possible values. When transmission size is small, time to fixation values are closer to the case where the within-host population size is reduced to match the number of pathogens transmitted (Figure 22, dark gray). As the infection duration shortens, the number of generations the pathogen experiences a bottleneck increases. As a result, the time to fixation is expected to approach this lower limit. On the other hand, larger transmission sizes are much closer to the time to fixation when the transmission size is equal to the current intrahost population size (Figure 22, light gray). Instead of reducing the severity of the bottleneck, another way to diminish its effect is to reduce the frequency of bottlenecks. Longer infection times will result in a smaller number of transmissions until fixation, thereby weakening the effect of transmission bottlenecks.

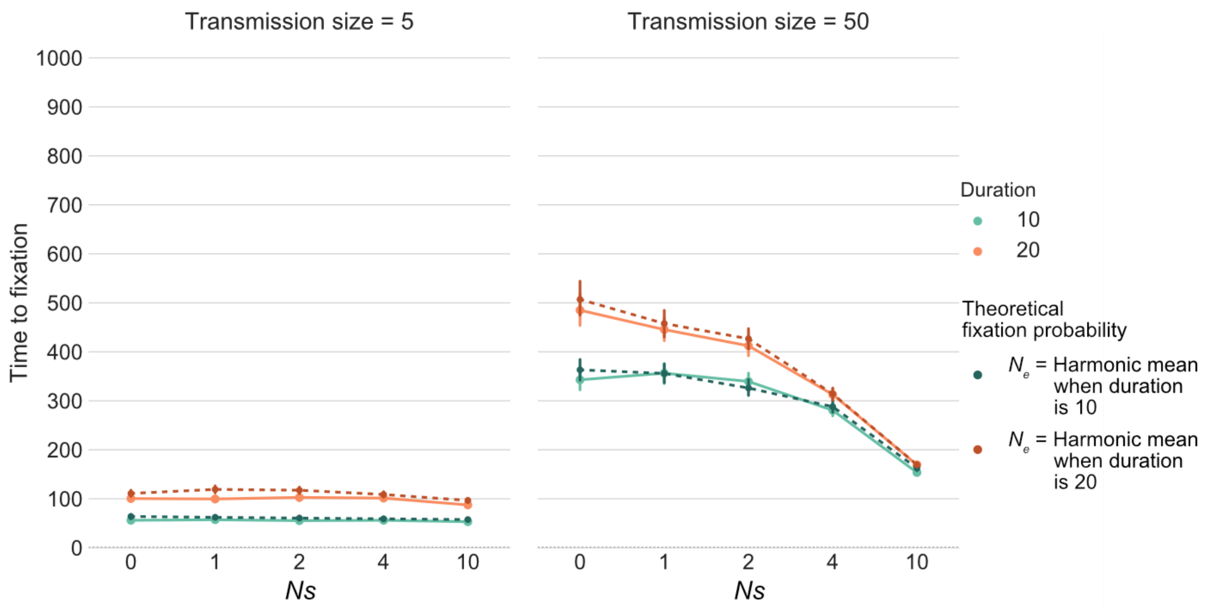


Figure 23. Time to fixation conditioned on fixation of the infection-transmission process depends on the harmonic mean of the sample sizes over time.

Time to fixation given fixation for different scaled selection coefficients when 5 pathogens – equivalent to 1% of the within-host population – (left) or 50 pathogens – equivalent to 10% of the within-host population – are transmitted between infections. Green and orange points indicate the length of time until fixation when the infection lasts for 10 or 20 pathogen generations respectively. Dark green and dark orange points indicate the time to fixation using the harmonic mean of within-host population size and respective transmission sizes in an infinite amount of time.

I also tested whether the harmonic mean of the fluctuating population sizes is a good approximation to compute the time to fixation conditioned on fixation of the allele of interest. Here, I compared the length of time to fixation between a fluctuating pathogen sample size due to recurrent infection and transmission, and a constant pathogen sample size given by the harmonic mean of within-host population size and transmission size (Figure 23).

Results from simulations using the harmonic mean show a good fit with the periodically fluctuating case. This confirms that recurrent infection and transmission acts to reduce the effective population size of the pathogen, and as a result, decreases the efficacy of selection.

Shape of the pathogen transmission tree and the number of observed fixations

In this experiment, I analyzed the role of that disease spreading plays in the fixation of mutations. Specifically, I examined whether the shape of the transmission tree influences the frequency of mutations. I wanted to know whether particular types of transmission trees promote, or hinder fixation compared to others. I considered three tree topologies that all start from a single infected host and expand to eight infected hosts after each transmission chain experiences three transmission events; but differed in the number of new infections each infected host can produce. After 1000 pathogen generations and 1000 independent realization, tree models consistently showed higher levels of fixation compared to their unstructured counterparts (Figure 24).

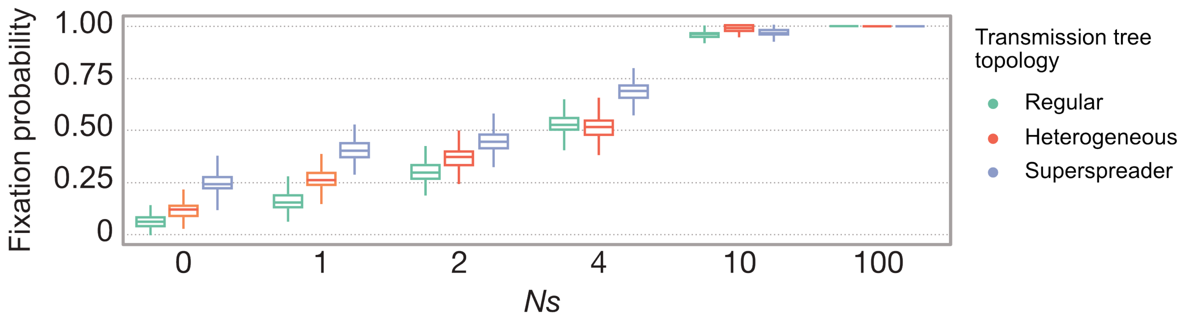


Figure 24. Probability of fixation on tree structures after 1000 pathogen generations.

For every scaled selection coefficient (Ns), different tree topologies, as shown in Figure 19, were compared with each other.

Between tree topologies, the superspreader tree created the greatest divergence from the null model. On the other hand, the binary tree closely resembled the unstructured model. Null models also differed in fixation probabilities confirming that the rate of population expansion affects the probability of observing fixation conditioned on time. However, the magnitude of the differences observed between tree topologies do not match and is much greater compared to the differences between null models. This indicates that the observed differences on fixation probability result from network effects rather than the underlying increase in population size.

A comparison between tree topologies showed that the superspreader tree have significantly higher fixation probabilities after 1000 generations compared to a regular binary tree and a heterogenous tree. Though fixation probabilities are all higher across scaled selection coefficients, fixation under neutral and weak selection appear more affected compared to stronger levels of selection. Unfortunately, given that these observations are conditioned at a particular time, it cannot be determined whether the heightened fixation probabilities are caused by (1) a faster time to fixation or (2) a decrease in the ultimate fixation probability.

A side-effect of placing pathogens into a tree structure is the overall pathogen population becomes structured. Each infected host in the transmission tree becomes an isolated population of pathogens that have no opportunity to mix. This creates divergent lineages with independent evolutionary outcomes. For any two hosts with an initial pathogen frequency p at the beginning of infection, isolated lineages mean that each host will most like have different frequencies by the end of the infection. As a result, one genotype could reach fixation in one host and another genotype can become fixed in the other host. This dynamic prevents the eventual fixation of a single genotype throughout the entire pathogen population.

DISCUSSION

The role of the duration of infection is an important parameter in epidemiological models of outbreaks and epidemics. In these models, the duration of infection is given by the reciprocal of the rate of recovery γ, which indicates the rate at which the supply of infected individuals decreases in the population. In this study, I showed that the duration of infection is also an important factor that affects the population genetics of pathogens. As pathogens replicate only when it infects a host, pathogen evolution only occurs during this time. Infectious diseases have vastly different infection times, some viral infections occurring only for days, while other infections are chronic and last throughout the lifetime of the host (Fields et al., 2013). Between acute and chronic infections, very different patterns of evolution have been observed. In chronic infections, pathogen populations exist for a long time within the host which results in significant within-host adaptation. Evolutionary changes of within-host populations of pathogens are especially evident during drug treatments where pathogens have been observed to develop drug resistance in a short period of time. In acute cases of infection, much less is known about the effects of the duration of infection on the evolution of pathogens.

These results indicate that the duration of infection and transmission bottlenecks are key factors that determines the evolution of pathogens within hosts. In acute infections, the short duration of infection prevents significant changes to pathogen genotype frequencies before the infection is cleared or transmitted. In my simulations, I assumed that transmission of pathogens occurs at the end of the duration of infection which at first appears to be a very unrealistic assumption. Transmission of pathogens tend to occur when viral titer is at its peak. At this time point, the population of pathogens with the host is at its peak and coincides with the time the symptoms of the disease, which facilitate transmission of pathogens, are strongest. Thus, the actual number of generations that acute infection pathogens have is significantly shorter that the observed duration of the infection. Given that acute viral infections last only for a few days and the number of lytic cycles that replicate the pathogen are few, my model's assumption represents the maximum opportunity to noticeably change the frequency of genotypes present.

Another hindrance to beneficial mutations is the presence of periodic bottlenecks resulting from transmissions between hosts. While a rigorous approximation of the average number of individual pathogen that typically creates an infection is unknown, the number is expected to vary depending on the mode of transmission and inoculation. The independent action hypothesis of infection states that each infectious pathogenic unit has a non-zero probability of infection and that pathogen units act independently of other units. Tests of the independent action hypothesis on plant and animals have largely supported these claims (Zwart et al., 2009; Zwart & Elena, 2014). This means that the lower limit of transmission is a single infectious pathogen and represents an extreme case of population bottleneck. Under this condition, transmission immediately causes fixation of a genotype in the new host. My model considers two cases of transmission bottlenecks that represents a 1% and 10% reduction in population size. Population bottlenecks have been known to rapidly decrease genetic variation in populations. Assuming that bottlenecks caused by transmissions sample pathogens randomly, then the probability of sampling a genotype becomes frequency-dependent and contingent on the its fitness. This reduces the chance of infrequent variants from being selected. Since new mutations start at a frequency equal to the reciprocal of the pathogen population size, there is an extremely low chance of being transmitted and spreading over the host population. Even given some time to increase in frequency, simulations showed that a 1% bottleneck significantly reduced the effect of selection on pathogen populations under recurrent bottlenecks. Moreover, when a mutant does reach fixation, the number of generations it took to reach fixation was insensitive to the value of the scale correlation coefficient. This strongly suggests that the presence of successive bottleneck events facilitated fixation rather than being mediated by intrahost selection.

The combination of short duration times and the effect of severe population bottlenecks could explain why naturally-occurring positive selection is rarely observed within hosts, and positive selection detected from non-intrahost samples may be an artifact of the transmission process. The transmission tree model showed that within-host fixation is more like to occur than overall fixation across all hosts. In this scenario, within-host evolution will rarely affect overall frequencies and the shape of the transmission tree dictates the polymorphism in the host population. This suggests that the distribution of genotypes over a host population is more likely a result of the transmission process than selection for a biological function. True positive selection in viral pathogens would only occur when the effect of mutations are extremely large to overcome the short infection cycles, or possibly as a result of a selected site experiencing independent mutations across different host infections.

CHAPTER 5
VIRUS EVOLUTION IN ALTERNATING HOSTS: FIXATION AMIDST SHIFTING FITNESS LANDSCAPES

INTRODUCTION

Compared to DNA viruses, RNA viruses have elevated mutation rates because of the high error rate of RNA-dependent RNA polymerases that replicate RNA virus genomes. However, Dengue virus and other vector-borne RNA viruses appear to evolve slower compared to RNA viruses that exclusively infected one or a closely-related group of host species (Ciota & Kramer, 2010). One explanation for this discrepancy implies that the large taxonomic difference between the host and vector species (often vertebrate and arthropod, respectively) create very different environments that prevent the pathogen from reaching the optimum fitness for either host species.

Called the trade-off hypothesis, this model predicts specialization by virtue of adaptation to one host species results in maladaptation in the other due to the different host environments. To remain viable in both host species, vector-borne pathogens have to find a compromise in the divergent host fitness landscapes. As a result, pathogens alternately infecting two hosts have a reduced fitness compared to pathogens that replicate only in a single host environment. If the alternating infection cycle does impose a compromise, then breaking the alternating cycle and solely infecting one or the other host should result in increased fitness for that particular host. Conversely, according to this hypothesis, any change conferring a fitness increase in one host should be accompanied by a fitness decrease in the bypassed host. Experimental evolution has indeed shown that continuously infecting vector-borne pathogens in a single host species does increase its fitness compared to an alternating cycle (Ciota & Kramer, 2010; Deardorff et al., 2011; Novella, Presloid, Smith, & Wilke, 2011; Vasilakis et al., 2009). However, the evidence for a concomitant fitness decrease in the bypassed host species has been inconsistent (Deardorff et al., 2011; Vasilakis et al., 2009). Instead, changes that caused an increase in fitness in one host appeared to have no effect on fitness in the other host or was also beneficial. Based on these results, it is likely that the host fitness landscapes share at least some peaks and valleys despite the large taxonomical distances separating natural hosts and vectors.

In this chapter I explore the role of alternating host cycles on the short-term evolution of viral pathogens using computational simulations. Under this theoretical framework, I model the protein evolution of the Dengue virus envelope protein under realistic parameter values as it evolves in human and mosquito hosts. I consider scenarios where conflict does and does not exist, as well as different magnitudes of selection antagonism. I show that alternation slows down evolution due to clonal interference caused by linkage between sites. In this model, conserved sites could be caused by strong purifying selection in either host and vector species, or by purifying selection in both hosts but acting on different sets of sites.

METHODS

Sequence alignment

Dengue virus envelope amino acid sequences were compiled and retrieved from the UniProt database using the BLASTP search algorithm. Using the New Guinea C Dengue virus type 2 envelope sequence as the reference (UniProtKB: P14320), I looked for the first 1000 closely-matching sequences that matched the reference up to an E-value of 10.

To determine conserved and variable sites, I constructed a multiple sequence alignment using MUSCLE (Edgar, 2004) (using parameters -maxiter 2 -diag). Using custom Python scripts, I cleaned up the alignment by the removing flanking regions before and after the envelope sequence and removing sites that contained gaps at any sample and from the filtered set of amino acid sites. This process generated an alignment with 495 amino acid sites, of which 143 sites were variable.

Among the amino acid variants at each variable site, the top variant by frequency was identified and designated as the major allele for the particular site. The identity of the major allele at each site was later used to create the fitness matrix in the simulation.

Simulating periodic host alternation

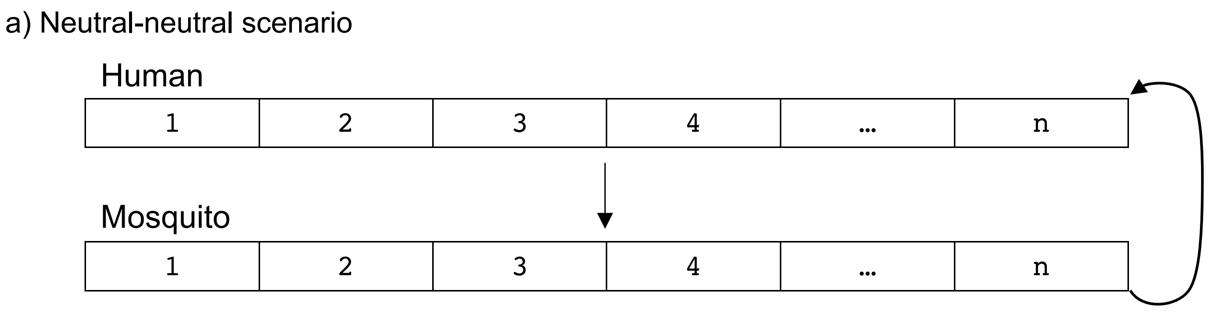
Vector-borne pathogens alternately infect two different species of hosts. Here, I simulated the transmission dynamics of Dengue viruses between humans and mosquitos to examine the effects of host alternation on fixation and genetic variation. I simulated this periodic alternation using Contagion, a forward genetic simulator I developed and previously discussed in detail in chapter 2. Briefly, this simulation framework enables hosts to be grouped into distinct host types such that host-specific properties can be assigned to each group and transmission can be restricted to occur between different host species. In this case, I created a linear bipartite network containing two host types representing humans and mosquitos. In a bipartite network, connections between host occur only if it is between different host types. By this rule, it is impossible to directly transmit between hosts of the same type (either both human or both mosquito). This ensures that pathogen populations are always alternately infect different host species and are subjected to alternating environments.

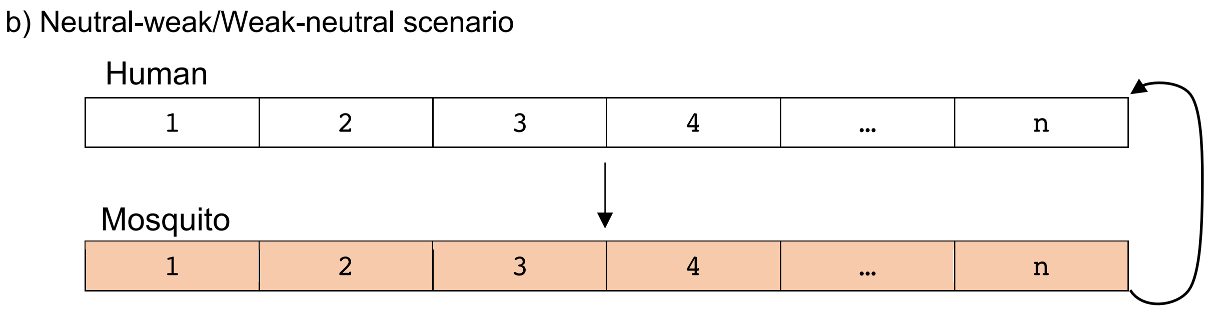
To simulate transmission, 10 pathogens or 1% of the pathogen population was randomly sampled without replacement from the existing intrahost population and was transferred into the new host to form the founder population. To simplify the epidemic simulation, I fixed the duration of infection to 10 or 20 generations assuming that one lytic cycle to produce new viruses lasts one day and that the average infection time is 2 weeks of 14 days. Moreover, I removed stochastic effects on transmission of the pathogen by setting the transmission probability to 1.0 to ensure that the transmission chain length is equal across simulation instances. Finally, all simulations were performed for 1000 pathogen generations and replicated 100 times.

To construct the genetic simulation, I modeled the 495 amino acid sites in the envelope protein of Dengue virus type 2 assuming that each site independently contributed to overall fitness and all sites are completely linked. For the transition rate matrix, I used the BLOSUM62 substitution matrix and conditioned it to reflect the transition probabilities of changing to another amino acid given that a substitution occurs. To model selection, I used a fitness matrix method that specifies the fitness contribution of each amino at each site. I tested several scenarios ranging from neutral evolution to extreme preference for a particular allele. When simulating fitness differences, the most frequently occurring amino acid present in the site (based on sequence alignment) was given a fitness of 1.0 while all the other alleles were assigned a fitness of $1 - s$, where s is the selection coefficient. Initially, the simulation was created by seeding 10 pathogens that have the same genotype that contained all preferred states across all sites in the sequence. The mutation rate was set to 1×10^{-5} substitutions/site/generation and the pathogen population per host was set to $N_d = 1000$.

I examined the different possible selection regimes operating during host-vector infections. First, I considered the completely neutral case where no fitness differences existed among the 20 different amino acids in both humans and mosquito hosts. To construct human and mosquito fitness matrices, each potential amino acid state at every site was given a fitness of 1.0 such that no fitness differences among all the possible amino acid combinations and no host-dependent conflict is present (Figure 25). Under this model, amino acid changes are not penalized as they do not change the fitness of the pathogen. Thus, this scenario represents evolution under completely random circumstances.

Next, I simulated the effect of purifying selection occurring in one of the two host species. This model is a simplification that mimics the behavior of experimental evolution studies that showed that one host imposed a slower evolutionary rate compared to the other host (Bedhomme, Lafforgue, & Elena, 2012; Coffey et al., 2008; Deardorff et al., 2011). I considered three scenarios based on where purifying selection is present: host only, vector only, and host and vector. In all these scenarios, each site has one preferred amino acid variant with a fitness of 1.0 and 19 equally unpreferred variants with a fitness of $1 - s$. The most frequency variant in the alignment of Dengue virus envelope sequences was considered as the preferred amino acid variant for that particular site. For each scenario, I simulated a weak selection case where the scaled selection coefficient $Ns = 1$ and a strong selection case where $Ns = 10$ (Figure 25). I also simulated cases where weak or strong selection was present in both hosts.





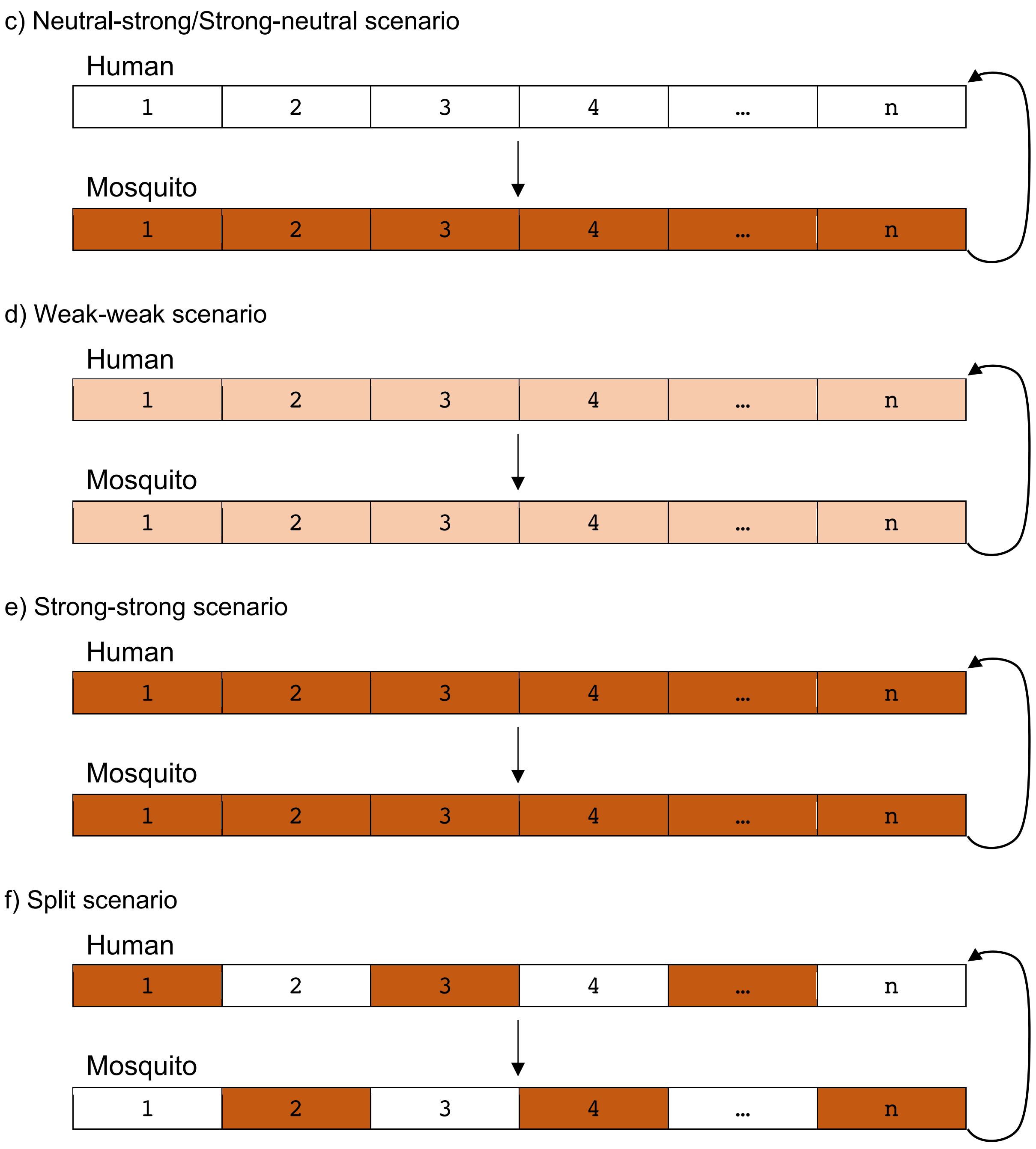


Figure 25. Visual representations of selection regimes tested under the alternating host model.

Each box represents a site. Intensity of shading indicates the strength of selection on the site, where no shading means no fitness differences exists for alleles of that site.

I also look at the possibility that purifying selection observed in the evolution of vector-borne pathogens resulted from selection present in both host and vector but acting on different sites. In this model, one half of the sites are under selection in the human host while the other half is under selection in the vector (Figure 1F). To create this "split" case, sites were split into two groups by odd or even positions. Odd-numbered sites were under selection during human infection but evolved neutrally during mosquito infection. Conversely, even-numbered sites evolved neutrally during human infection, but were purifying selection during mosquito infection. I simulated these relationships under three different strengths of selection – weak (*Ns* = 1), strong (*Ns* = 10), and very strong (*Ns* = 100).

To quantify the effect of shifting selection pressures, I measured the number of new mutations that went to fixation of pathogens and traced the evolutionary history of fixed and segregating mutations. In all simulations, the original genotype is the most fit genotype and new mutations may decrease the pathogen's fitness. Under neutral evolution, no fitness differences exist between the different amino acids and new mutations do not adversely affect pathogen fitness. On the other hand, when fitness differences exist between alleles, purifying selection acts to impede the fixation of new mutations. A new mutation has been fixed when 1) only one type of amino acid is observed in the particular site, and 2) the observed amino acid is different from the original genotype. Intrahost fixations were counted by comparing the site-by-site identity of sequences found after 1000 generations with the original genotype.

RESULTS AND DISCUSSION

Previous studies have examined the role of host alternation in the evolution of pathogens using *in vitro* and *in vivo* approaches. To simulate pathogen evolution *in vitro*, pathogens are inoculated into a cell culture derived from one of its host species and incubated to simulate replication within the host. During this step, pathogens could be passaged into fresh cell cultures as long as it remains of the same host type. After a certain time, viruses are harvested from cell culture fluid and inoculated into a different cell line to simulate transmission between host species. This completes one infection cycle in a single host. The process of inoculation, incubation, and transfer is repeated until the desired number of infection cycles and transmissions has been reached. The process is similar for *in vivo* approaches except that live animals are used instead of cell lines. Using *in vivo* models enables capturing the dynamics of local and systemic viral infection, as well as cell tropisms and tissue-specific infections at the cost of more complicated experimental design and less control of variables that may affect pathogen evolution. Given the numerous variables that jointly affect pathogen evolution, isolating the individual contribution of processes in live models is challenging because extraneous variables must be tightly controlled, and jointly contributing parameters must be held constant.

Probabilistic models and simulation offer an alternative method to examine pathogen evolution with complete control over the environment and the contributing processes. There are three key advantages of *in silico* models over live models of evolution: absolute control of parameters, repeatability, and data resolution. However, computer simulations are limited by our ability to capture evolutionary dynamics into mathematical equations and computer language. Thus, a certain amount of realism is lost when computer models are used.

In this chapter, I am interested in the effect of host alternation on the evolution of pathogens. Host alternation presents a unique evolutionary problem for vector-borne pathogens because alternately reproducing between two different environments may constrain the evolution of the pathogen. The host alternation problem can be rephrased as a problem of alternating fitness landscapes. Given these two environments, the fitness landscapes of these environments could be very similar to each other such that the landscapes share fitness optima and minima. On the other hand, the fitness landscapes could be vastly different the fitness optima in one environment is different from the optima in the other environment. In the extreme case, the fitness optima of one landscape may even coincide with the fitness minima in the other. Finally, the magnitude of changes in the fitness landscapes may be very disproportionate such that the effect of one landscape dominates over the other. This means that one host environment is more permissive to change because it has a less jagged landscape while the other host environment is quite restrictive due to the high peaks and deep valleys in its landscape.

Experiments have suggested that pathogen evolution may be most similar to the last case where there is relaxed selection in one host and strong selection in the other host. Here, I modeled three fitness landscape scenarios to examine the effect of shifting landscapes on the fixation of new mutations. First, I considered the random scenario where fixation is completely driven by genetic drift and random sampling during transmission to establish a null model of pathogen evolution in alternating hosts. Then I simulated evolution where the fitness landscapes were consistent between hosts, and where one landscape was dominant over the other. I implemented these landscape configurations by simulating the infection cycle of Dengue viruses between humans and mosquitos using the individual-based models implemented in Contagion. Under this model, humans and mosquitos are represented as nodes in the network and connections between nodes indicates the potential for transmission. To alternate infections between host species, connections only occur between a human node and a mosquito node such that pathogens have to alternately infect hosts after every infection cycle. This creates a special kind of network called a bipartite graph.

In the null model, also known as the neutral-neutral scenario, amino acid changes do not affect the fitness of the pathogen whether it is replicating within humans or mosquitos. Thus, changes to the frequency of an allele or a genotype happened due to random chance or due to or random sampling during transmission. No allele or genotype has any fitness advantage. After 1000 pathogen generations and 99 transmissions, an average of 4.37 new mutations were fixed in the simulation and an average of 19.10 sites remained polymorphic (Table 3). As previously discussed in chapter 3, the fixation probability of pathogen mutations is a composite value dependent on the strength of selection, infection duration and transmission frequency. This means that the fixation rate of new mutations under this model is not only based on the rate of new mutations entering the population, but also is affected the bottleneck size and frequency of transmissions. According to population genetics, the fixation rate in a haploid population with a constant population size N evolving under neutral evolution is equal to product of the number of new mutations per generations $N\mu$ and the fixation probability of a new mutation $\frac{1}{N} \times N\mu = \mu$. Given a mutation rate of 1×10^{-5} substitutions/site/generation and the simulation having modeled 495 sites for 1000 pathogen generations, this equation predicts that there should be 4.95 fixations during the span of the simulation. And yet, simulation results show an average of 4.37 new mutations were fixed at the end of the simulation. Note that as I discussed in chapter 3, the recurrent demographic changes inherent to the pathogen life cycle changes the fixation probability of a new mutation. In this case, the fixation rate under neutral evolution is only proportional to simulation result by a factor *a* such that $a\frac{1}{N} \times N\mu = a\mu$.

Another key difference is the fact that the sites in this sequence are completely linked to each other. When sites are completely linked, no recombination takes place between sequences and the entire sequence acts as a linkage block. As a consequence, the fate of new mutations becomes tied to the fate of other mutations present in the linkage block and the rate of fixations is less than the expectation under free recombination.

Next, I considered the case where purifying selection occurs in one of the two hosts. Consider the cases where weak purifying selection (*Ns* = 1) occurs in human hosts only, or in mosquito hosts only (Table 3). Under these scenarios, mutations were free to accumulate and increase in frequency without any effect on fitness while the pathogen is replicating in one host. However, when the pathogen population is transmitted to the other host species, mutations that previously had no effect on replicative fitness now are deleterious. Pathogens that harbor mutations now have lower replicative fitness and are less likely to be found in the next generation of pathogens. Thus, purifying selection present in at least one of the hosts should decrease the rate of fixations compared to the null model. This model tries to replicate the experimental evolution studies of Vasilakis et al. (2009) where they observed vectors constrain the evolution of Dengue viruses *in vitro* and Deardorff et al. (2011) that found that natural bird host slows the evolution of viral pathogens.

Table 3. Fixation and polymorphism of new mutations under the alternating host model.

Class	Ns_{host}	Ns_{vector}	Intrahost Fixation (# sites)	Intrahost Polymorphism (# sites)
Neutral-neutral	0	0	4.37	19.10
Weak-neutral	1	0	4.00	18.97
Neutral-weak	0	1	4.00	18.38
Weak-weak	1	1	4.20	17.70
Weak-weak	0.5	0.5	4.03	18.67
Strong-neutral	10	0	3.71	18.67
Neutral-strong	0	10	3.58	18.18
Strong-strong	10	10	3.16	18.03
Strong-strong	5	5	3.71	18.38

Because pathogen genomes are completely linked, there is effectively no recombination occurring between viral sequences, which decreases the efficiency of selection. The lack of recombination in pathogen genomes make them susceptible to clonal interference and Muller's ratchet. Clonal interference occurs when a beneficial mutation become extinct due to competition with a new beneficial mutation occurring at a different site in sequence. If recombination is present, the two beneficial mutations could find themselves in the same sequence and both could go to fixation more rapidly than if they were separated, assuming there is no negative epistasis between the two. However, when recombination is not present, then the second mutation has to occur in a sequence where the first mutation is already present for both to reach fixation. Otherwise, the two mutations would compete with each other and could result in one of the mutations going to extinction. Conversely, if recombination was present, deleterious mutations could be more efficiently removed from the population because its fate is not tied to other variants in the sequence. The lack of recombination enables deleterious mutations to go into fixation at a high rate than expected especially when advantageous mutations are present in the background. Linkage decreases the efficiency of selection in pathogens and creates an interesting dynamic in the case of alternating environments.

When selection is present in one of the hosts, the number of fixations observed after 1000 pathogen generations decreased as expected compared to neutral evolution (Table 3). Since the fixation probability is proportional to a factor of the fixation rate, then the lower fixation probability of deleterious mutations decreases its fixation rate. However, in this scenario, selection is not constant over time but is alternately present or absent as the pathogens consecutively replicate in permissive and stringent host environments. In cases where selection fluctuates, the time scale of environmental change is a critical factor in determining the fixation probability of new mutations. If the environment changes rapidly relative to evolutionary time, then the fixation probability is determined by the average effective fitness effect over time in some cases (Mustonen & Lässig, 2009). In general, environmental changes have been found to reduce the efficiency of selection on a single locus (Cvijović, Good, Jerison, & Desai, 2015). Under complete linkage, the number of fixations in the alternating selection regime appears to behave similar to the time-averaged selection coefficient. In both weak and strong cases, the number of fixations when selection is present only in one of the hosts is similar to the observed number when selection is present in both hosts at half the value (Table 3).

Comparing the deleterious fixation rates between the standard population genetics model and the simulation, results showed an increasing divergence as the scaled selection coefficient increased. At $Ns = 1$, the fixation rate of deleterious mutations is comparable between the standard and the pathogen evolution models. However, as the strength of purifying selection increased, the simulation seemed to be largely unaffected by the parameter change (Figure 25). Simulation results were robust to changes in selection coefficient because the effect of repeated population bottlenecks dominated the evolution of pathogen at these parameter values. As discussed in chapter 3, the duration of infection time and the frequency of transmissions are key factors that affect the fixation probability of an allele. As infection duration shortens or as frequency of transmissions increases, the overall fixation probability approaches the neutral expectation. Thus, between *Ns* = 1 and *Ns* = 10, the force of selection is subdued by the random effects brought by severe bottlenecks during transmission. If true, this suggests that viral pathogen evolution is more a consequence of random events than controlled by deterministic forces such as selection.

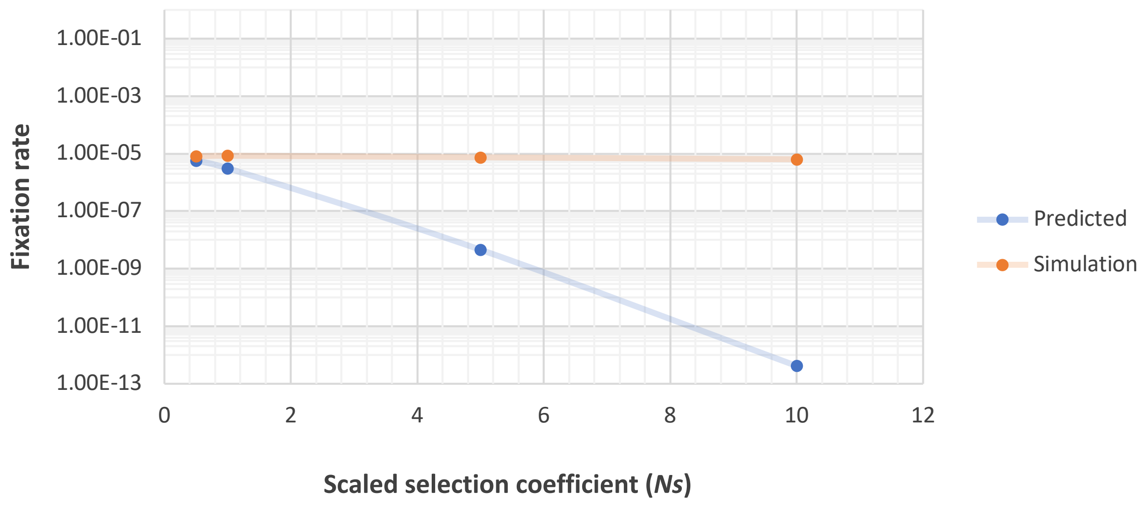


Figure 26. Divergence in fixation rates of new mutations between the standard population genetics model and the pathogen evolution network model.

The close range of simulation results may also have been caused by the low supply of mutations as a result of using realistic parameter values to simulate Dengue virus molecular evolution. While this helps account for realism in the simulation, the drawback of this approach is the rate of new mutations could be insufficient to produce the necessary number of fixations to create a powerful enough statistical comparison. One way to increase the frequency of observed fixations is to increase the number of generations simulated. However, as the number of generations increase the amount of time the simulation runs, it is not always a viable solution. For instance, the current set-up simulates 1000 pathogen generations and takes approximately 600 seconds or 10 minutes to finish a single instance of the simulation and replicating it 100 times takes 60,000 seconds or about 17 hours. To increase the number of recorded fixations 10-fold would require on average a 10-fold increase in simulation time as well, increasing the simulation time to 600,000 seconds or approximately 7 days. The alternative is to increase the rate of mutation to increase the supply of mutations, and thereby increase the probability of observing fixations. However, this means adopting an unrealistic parameter value for the pathogen being modeled. Both approaches are currently being explored in order to increase the power of statistical tests to compare these models.

I also investigated the possibility that purifying selection occurs cooperatively between hosts to conserve different sites. Instead of all sites undergoing purifying selection in one host and freely changing under neutral evolution in the other host, I consider the possibility of a “split” scenario where one host species dictates the evolution for a particular set of sites while the other host species predominates the evolution for another non-overlapping group of sites (Table 4). This scenario would also result in effectively half the original fitness difference similar to the alternating scenario. For example, there are two groups of sites A and B that are completely linked in a genotype (Figure 25f). In host 1, site A is under purifying selection while site B evolves neutrally. On the other hand, in host 2, site A evolves neutrally while site B is under purifying selection. Within host 1, only changes to site A affects it replicative fitness and site B is free to mutate without consequence. However, once the pathogen infects host 2, site B variants that previously did not have an effect on fitness are now subjected to selection. Site A, which was under selection in host 1, is now free to change without having an effect on pathogen fitness. Under this scenario, the effective selection coefficient would be the average of the two hosts and should have the same effect as cases that have the same effective fitness effect value.

Table 4. Comparison of fixation and polymorphism of new mutations between the split and alternating host models.

Class	Ns_{host}	Ns_{vector}	Intrahost Fixation (# sites)	Intrahost Polymorphism (# sites)
Neutral-neutral	0	0	4.37	19.10
Complete	1	1	4.20	17.70
	10	10	3.16	18.03
Alternating	1	0	4.00	18.68
	10	0	3.64	18.43
Split	1*	1*	4.22	19.13

	10*	10*	3.41	18.70
	100*	100*	1.98	16.72

* odd-numbered sites were subjected to purifying selection in human hosts while even-numbered sites were under purifying selection in vector hosts. The intensity of selection is indicated by the respective scaled selection coefficients listed in the table.

Overall, the results of the split scenario were not consistent with alternating host model. It was expected that splitting the action of selection would not affect the evolution of new mutations as the average fitness effects for alleles at each site remained constant. In the case of alternating selection, the preferred allele at all sites is advantageous or neutral while the unpreferred alleles are either disadvantageous (0.9) or neutral (1.0). Thus, in one host, all the sites are always under selection, while in the other host, all the sites are neutrally evolving. On the other hand, in the split scenario, the preference and fitness of alleles remained the same except that only half of the sites were under selection in each host. This means selection was present in both hosts but acts on different sites. Based on the simulation results, the split scenario lied in between the case where selection was present in alternating infections, and when selection was consistently present in both hosts. Since demographic factors known to affect fixation were held constant, this behavior suggested that the linkage among sites in the pathogen sequence alters fixation dynamics of new mutations. This aspect requires further investigation to explain this discrepancy. A future study could look at the differences between the alternating "complete" and "split" scenarios with differing rates of recombination to determine whether linkage is the cause of this unexpected result.

CHAPTER 6
CONCLUSION

Pathogen evolution is a far from a settled topic because there are many variables that can affect its outcome. The complexity of pathogen evolution derives from its unique life cycle characterized by a never-ending series of infection and transmission which ties pathogens to their host and are affected by the host's behavior and interactions. One clear example of this is how hosts can interact with each other to create favorable routes of transmission. For HIV and other sexually-transmitted infections, this may be the choice of sexual partners and the frequency of sexual interactions. For Influenza, this may be the person's choice of insisting on going to school or work as opposed to resting until the symptoms disappear. Although pathogens take advantage of hosts in order to reproduce, if it cannot encounter a new host to infect, it faces a dead end, imprisoned within the host. Thus, the host plays a significant role in determining the pathogens ultimate fate.

To understand how pathogens evolve requires an accurate depiction of the essential qualities that creates this dynamical system. Although there are multitudes of epidemic models available to study the spread of diseases that pathogens cause, there are few the genetic models that can recapitulate the evolutionary history of pathogens without oversimplifying it. Through this project, I developed a model for pathogen evolution by embedding the evolutionary process within a network. Based on this model, I showed that the coupling of evolutionary processes and epidemiological dynamics forms a feedback loop. While the effect of evolutionary changes on epidemiological processes has been well documented already, there is little evidence to suggest that the converse relationship – that epidemiological factors can tilt evolution – is true. However, it is intuitive to consider these processes as affecting each other. From a population genetics perspective, evolution occurs because of changes to the frequency of genetic information in the form of alleles. The great mystery in population genetics is deriving the factors that drive the change, if any at all. Expanding this perspective from the processes that occur within the host to include the overall population of hosts, a simple explanation based on probability can be derived to prove that the structure of host populations affects the molecular and phenotypic evolution of pathogens. By imaging the host population as a connected network, then connections between hosts create paths to traverse the network. From the collection of all possible paths, some paths increase the probability of spreading while other and paths lead to dead ends. Therefore, if a new mutation arises in a pathogen population located in a well-connected part of the host network increases it chance of survival compared to a new mutation that emerge in an infection found in a sparsely connected region even if the mutations share the same fitness. Therefore, the eventual fate of new mutations is governed by forces within the host and by interactions between hosts.

Within-host studies have found little evidence to support the idea that positive selection of pathogens naturally occurs within hosts. However, the signature of positive selection has been detected in viruses and other pathogens from samples taken from different individuals. Given these conflicting claims, it is interesting how we can reconcile this problem. In this study, I attempted to untangle these conflicting observations the network model of pathogen evolution I devised. Due to the periodic infection and transmission pathogen lineages undergo, I found that these events dampen the effect of natural selection even at pathogens possess a large fitness. Two main factors impeded the effect of positive selection in pathogen evolution. First, the time it takes for selection to significantly raise the frequency of a new mutation is too long compared to the duration of acute infections. This means that new mutations remain at low frequency throughout an infection unless the fitness advantage is extremely high, or the duration of the infection is significantly long. This explains why most findings of positive selection within hosts have been from pathogens that cause chronic infections. If beneficial mutations exist at low frequencies, the probability that these mutations are transmitted is slim, especially when the number of infectious units transmitted is low. The population bottleneck that occurs every transmission is the second reason positive selection cannot operate efficiently. Experimental studies have shown that transmissions tend to impose a harsh bottleneck on pathogens. Since pathogens transmits periodically, this means that bottlenecks recurrently occur over a short period of time. My results indicate pathogen evolution is not sensitive to fitness differences as the fixation probability of mutations tends to be flat up to moderate level of selective advantage. This indicates that the periodic expansion and reduction in population size in tandem with the short intervals between bottlenecks inhibit the role of positive selection in pathogens. This suggests that transmission parameters could play a bigger role in the evolution of pathogen than natural selection.

Placed in the context of a host network, both the density and the structure of the network influences the observed evolution of pathogens across the host population. Results from simulations on networks showed that diversity is significantly reduced when the host network is sparsely-connected than when it is densely-connected. However, any type of network used to condition the potential paths of transmission always showed lower levels of diversity compared to an unstructured population. While the network model saw reduced diversity, fixation probability is also diminished because the network structure promotes differentiation between pathogen lineages. Networks therefore make it hard for an advantageous genotype to sweep the entire population unless it is related to host immunity. When the new mutation changes the pathogen's immunological profile such that is create a new serotype, then these kinds of changes are expected spread rapidly as the pathogen is granted an effectively immunological naïve host population to infect. It remains to be seen however, how we can explain other findings of positive selection that appear adaptive but are not immune-related.

REFERENCES

Abbey, H. (1952). An examination of the Reed-Frost theory of epidemics. *Human Biology*, *24*(3), 201–233. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/12990130

Akira, S., Uematsu, S., & Takeuchi, O. (2006). Pathogen recognition and innate immunity. *Cell*, *124*(4), 783–801. https://doi.org/10.1016/j.cell.2006.02.015

Albert, R., Jeong, H., & Barabási, A.-L. (2000). Error and attack tolerance of complex networks. *Nature*, *406*(6794), 378–382. https://doi.org/10.1038/35019019

Allen, L. J. S. (1994). Some discrete-time SI, SIR, and SIS epidemic models. *Mathematical Biosciences*, *124*(1), 83–105. https://doi.org/10.1016/0025-5564(94)90025-6

Anderson, R. M., Fraser, C., Ghani, A. C., Donnelly, C. A., Riley, S., Ferguson, N. M., … Hedley, A. J. (2004). Epidemiology, transmission dynamics and control of SARS: the 2002-2003 epidemic. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *359*(1447), 1091–1105. https://doi.org/10.1098/rstb.2004.1490

Anderson, R. M., & May, R. M. (1985). Vaccination and herd immunity to infectious diseases. *Nature*, *318*(6044), 323–329. https://doi.org/10.1038/318323a0

Anderson, R. M., & May, R. M. (1992). *Infectious Disease of Humans*. Oxford University Press.

Andersson, D. I., & Hughes, D. (1996). Muller’s ratchet decreases fitness of a DNA-based microbe. *Proceedings of the National Academy of Sciences of the United States of America*, *93*(2), 906–907. https://doi.org/10.1073/pnas.93.2.906

Andolfatto, P., & Przeworski, M. (2000). A genome-wide departure from the standard neutral model in natural populations of Drosophila. *Genetics*, *156*(1), 257–268. Retrieved from http://genomics.princeton.edu/AndolfattoLab/Publications_files/Andolfatto_Przeworski 2000.pdf

Antia, R., Ganusov, V. V., & Ahmed, R. (2005). The role of models in understanding CD8+ T-cell memory. *Nature Reviews Immunology*, *5*(2), 101–111. https://doi.org/10.1038/nri1550

Antia, R., Regoes, R. R., Koella, J. C., & Bergstrom, C. T. (2003). The role of evolution in the emergence of infectious diseases, *426*(December), 8–11. https://doi.org/10.1038/nature02177.1.

Bandín, I., & Dopazo, C. P. (2011). Host range, host specificity and hypothesized host shift events among viruses of lower vertebrates. *Veterinary Research*, *42*(1), 67. https://doi.org/10.1186/1297-9716-42-67

Barthélemy, M., Barrat, A., Pastor-Satorras, R., & Vespignani, A. (2005). Dynamical patterns of epidemic outbreaks in complex heterogeneous networks. *Journal of Theoretical Biology*, *235*(2), 275–288. https://doi.org/10.1016/j.jtbi.2005.01.011

Barton, N. H., Etheridge, A. M., & Véber, A. (2013). Modelling evolution in a spatial continuum. *Journal of Statistical Mechanics: Theory and Experiment*, *2013*(01), P01002. https://doi.org/10.1088/1742-5468/2013/01/P01002

Barton, N. H., & Whitlock, M. C. (1997). The evolution of metapopulations. In *Metapopulation biology* (pp. 183–210). Academic Press.

Batagelj, V., & Brandes, U. (2005). Efficient generation of large random networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, *71*(3), 1–5. https://doi.org/10.1103/PhysRevE.71.036113

Bedhomme, S., Lafforgue, G., & Elena, S. F. (2012). Multihost experimental evolution of a plant RNA virus reveals local adaptation and host-specific mutations. *Molecular Biology and Evolution*, *29*(5), 1481–1492. https://doi.org/10.1093/molbev/msr314

Beerli, P., & Felsenstein, J. (2001). Maximum likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(8), 4563–4568. https://doi.org/10.1073/pnas.081068098

Bergstrom, C. T., McElhany, P., & Real, L. A. (1999). Transmission bottlenecks as determinants of virulence in rapidly evolving pathogens. *Proceedings of the National Academy of Sciences of the United States of America*, *96*(9), 5095–5100. https://doi.org/10.1073/PNAS.96.9.5095

Beverton, R. J. H., & Holt, S. (1957). *On the dynamics of exploited fish populations. U. K. Min. Agric. Fish. food, Fish. Invest. (Ser. II).*

Bhatt, S., Katzourakis, A., & Pybus, O. G. (2010). Detecting natural selection in RNA virus populations using sequence summary statistics. *Infection, Genetics and Evolution*, *10*(3), 421–430. https://doi.org/10.1016/j.meegid.2009.06.001

Bull, J. J., Meyers, L. A., & Lachmann, M. (2005). Quasispecies made simple. *PLoS Computational Biology*, *1*(6), 0450–0460. https://doi.org/10.1371/journal.pcbi.0010061

Burch, C. L., & Chao, L. (2000). Evolvability of an RNA virus is determined by its mutational neighbourhood. *Nature*, *406*(6796), 625–628. https://doi.org/10.1038/35020564

Carvajal-Rodríguez, A. (2008). GENOMEPOP: a program to simulate genomes in populations. *BMC Bioinformatics*, *9*(1), 223. https://doi.org/10.1186/1471-2105-9-223

Cen, X., Feng, Z., & Zhao, Y. (2014). Emerging disease dynamics in a model coupling within-host and between-host systems. *Journal of Theoretical Biology*, *361*, 141–151. https://doi.org/10.1016/j.jtbi.2014.07.030

Centers for Disease Control and Prevention. (2012). *Principles of Epidemiology in Public Health Practice* (3rd ed.). U.S. Department of Health and Human Services.

Chao, L. (1990). Fitness of RNA virus decreased by Muller's ratchet. *Nature*, *348*(6300), 454–455. https://doi.org/10.1038/348454a0

Chare, E. R., & Holmes, E. C. (2006). A phylogenetic survey of recombination frequency in plant RNA viruses. *Archives of Virology*, *151*(5), 933–946. https://doi.org/10.1007/s00705-005-0675-x

Charlesworth, B. (2012). The effects of deleterious mutations on evolution at linked sites. *Genetics*, *190*(1), 5–22. https://doi.org/10.1534/genetics.111.134288

Chinese SARS Molecular Epidemiology Consortium. (2004). Molecular evolution of the SARS coronavirus during the course of the SARS epidemic in China. *Science (New York, N.Y.)*, *303*(5664), 1666–1669. https://doi.org/10.1126/science.1092002

Ciota, A. T., & Kramer, L. D. (2010). Insights into Arbovirus Evolution and Adaptation from Experimental Studies. *Viruses*, *2*(12), 2594–2617. https://doi.org/10.3390/v2122594

Codoñer, F. M., Darós, J.-A., Solé, R. V., & Elena, S. F. (2006). The fittest versus the flattest: experimental confirmation of the quasispecies effect with subviral pathogens. *PLoS Pathogens*, *2*(12), e136. https://doi.org/10.1371/journal.ppat.0020136

Coffey, L. L., Vasilakis, N., Brault, A. C., Powers, A. M., Tripet, F., & Weaver, S. C. (2008). Arbovirus evolution in vivo is constrained by host alternation. *Proceedings of the National Academy of Sciences of the United States of America*, *105*(19), 6970–6975. https://doi.org/10.1073/pnas.0712130105

Cotten, M., Watson, S. J., & Zumla, A. I. (2014). Spread, Circulation, and Evolution of the Middle East Respiratory Syndrome Coronavirus. *MBio*, *5*(1), . doi:10.1128/mBio.01062-13. https://doi.org/10.1128/mBio.01062-13.Editor

Crotty, S., Cameron, C. E., & Andino, R. (2001). RNA virus error catastrophe: direct molecular test by using ribavirin. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(12), 6895–6900. https://doi.org/10.1073/pnas.111085598

Crow, J. F., & Kimura, M. (1970). *An Introduction to Population Genetics Theory*. *Cours de l'University of Oslo Department of Informatics*. Blackburn Press. Retrieved from http://www.amazon.de/Introduction-Population-Genetics-Theory-James/dp/0060414383/ref=sr_1_3?ie=UTF8&qid=1321741251&sr=8-3

Cvijović, I., Good, B. H., Jerison, E. R., & Desai, M. M. (2015). Fate of a mutation in a fluctuating environment. *Proceedings of the National Academy of Sciences*, *112*(36), E5021–E5028. https://doi.org/10.1073/pnas.1505406112

Deardorff, E. R., Fitzpatrick, K. A., Jerzak, G. V. S. S., Shi, P.-Y., Kramer, L. D., & Ebel, G. D. (2011). West Nile virus experimental evolution in vivo and the trade-off hypothesis. *PLoS Pathogens*, *7*(11), e1002335. https://doi.org/10.1371/journal.ppat.1002335

Domingo, E. (2000). Viruses at the Edge of Adaptation. *Virology*, *270*(2), 251–253. https://doi.org/10.1006/viro.2000.0320

Domingo, E. (2002). Quasispecies theory in virology. *Journal of Virology*, *76*(1), 463–465. https://doi.org/10.1128/JVI.76.1.463-465.2002

Domingo, E., Sheldon, J., & Perales, C. (2012). Viral Quasispecies Evolution. *Microbiology* *and Molecular Biology Reviews*, *76*(2), 159–216. https://doi.org/10.1128/MMBR.05023-11

Duarte, E. A., Clarke, D., Moya, A., Domingo, E., & Holland, J. J. (1992). Rapid fitness losses in mammalian RNA virus clones due to Muller's ratchet. *Proceedings of the National Academy of Sciences of the United States of America*, *89*(13), 6015–6019. https://doi.org/10.1073/pnas.89.13.6015

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acid Research*, *32*(5), 1792–1797. https://doi.org/10.1093/nar/gkh340

Eigen, M., & Schuster, P. (1977). A principle of natural self-organization - Part A: Emergence of the hypercycle. *Naturwissenschaften*, *64*(11), 541–565. https://doi.org/10.1007/BF00450633

Erdős, P., & Rényi, A. (1959). On Random Graphs. I. *Publicationes Mathematicae*, *6*, 290–297.

Eubank, S., Guclu, H., Kumar, V. S., Marathe, M. V, Srinivasan, A., Toroczkai, Z., & Wang, N. (2004). Modelling disease outbreaks in realistic urban social networks. *Nature*, *429*(6988), 180–184. https://doi.org/10.1038/nature02541nature02541 [pii]

Felsenstein, J. (1974). The evolution advantage of recombination. *Genetics*, *78*(2), 737–756.

Fields, B. N., Knipe, D. M., & Howley, P. M. (2013). *Fields Virology, 6th Edition*. *Fields Virology*. https://doi.org/10.1093/cid/ciu346

Frank, S. A. (1996). Models of parasite virulence. *The Quarterly Review of Biology*, *71*(1), 37–78. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/8919665

Fujie, R., & Odagaki, T. (2007). Effects of superspreaders in spread of epidemic. *Physica A: Statistical Mechanics and Its Applications*, *374*(2), 843–852. https://doi.org/10.1016/j.physa.2006.08.050

Gaggiotti, O. E., & Smouse, P. E. (1996). Stochastic Migration and Maintenance of Genetic Variation in Sink Populations. *The American Naturalist*, *147*(6), 919–945. https://doi.org/10.1086/285886

Galton, F., & Watson, H. W. (1875). On the probability of the extinction of families. *Journal of the Royal Anthropological Institute*, *4*, 138–144.

Ganesh, a., Massoulié, L., & Towsley, D. (2005). The effect of network topology on the spread of epidemics. *Proceedings - IEEE INFOCOM*, *2*(C), 1455–1466. https://doi.org/10.1109/INFCOM.2005.1498374

Ghedin, E., Laplante, J., DePasse, J., Wentworth, D. E., Santos, R. P., Lepow, M. L., … St. George, K. (2011). Deep sequencing reveals mixed infection with 2009 pandemic influenza A (H1N1) virus strains and the emergence of oseltamivir resistance. *The Journal of Infectious Diseases*, *203*(2), 168–174. https://doi.org/10.1093/infdis/jiq040

Gladstien, K. (1977). Subdivided Populations: The Characteristic Values and Rate of Loss of

Alleles. *Journal Of Applied Probability*, *14*(2), 241–248. Retrieved from papers3://publication/uuid/913AF56E-D6BF-48F9-9C0A-B80E95D96D49

Gordo, I., Gomes, M. G. M., Reis, D. G., & Campos, P. R. (2009). Genetic diversity in the SIR model of pathogen evolution. *PLoS ONE*, *4*(3), 1–8. https://doi.org/10.1371/journal.pone.0004876

Goyal, S., Balick, D. J., Jerison, E. R., Neher, R. A., Shraiman, B. I., & Desai, M. M. (2012). Dynamic Mutation-Selection Balance as an Evolutionary Attractor. *Genetics*, *191*(4), 1309–1319. https://doi.org/10.1534/genetics.112.141291

Gregg, M. B. (2002). *Field Epidemiology*. Oxford University Press.

Grenfell, B. T., Pybus, O. G., Gog, J. R., Wood, J. L. N., Daly, J. M., Mumford, J. A., & Holmes, E. C. (2004). Unifying the Epidemiological and Evolutionary Dynamics of Pathogens. *Science*, *303*(5656), 327–332. https://doi.org/10.1126/science.1090727

Griffiths, R. C., & Tavaré, S. (1994). Sampling theory for neutral alleles in a varying environment. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *344*(1310), 403–410. https://doi.org/10.1098/rstb.1994.0079

Guillaume, F., & Rougemont, J. (2006). Nemo: an evolutionary and population genetics programming framework. *Bioinformatics*, *22*(20), 2556–2557. https://doi.org/10.1093/bioinformatics/btl415

Hagberg, A., Schult, D. a., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*.

Hall, J. P., Harrison, E., & Brockhurst, M. A. (2013). Viral host-adaptation: insights from evolution experiments with phages. *Current Opinion in Virology*, *3*(5), 572–577. https://doi.org/10.1016/j.coviro.2013.07.001

Haller, B. C., & Messer, P. W. (2017). SLiM 2: Flexible, Interactive Forward Genetic Simulations. *Molecular Biology and Evolution*, *34*(1), 230–240. https://doi.org/10.1093/molbev/msw211

Hartfield, M. (2012). A framework for estimating the fixation time of an advantageous allele in stepping-stone models. *Journal of Evolutionary Biology*, *25*(9), 1751–1764. https://doi.org/10.1111/j.1420-9101.2012.02560.x

Hernandez, R. D. (2008). A flexible forward simulator for populations subject to selection and demography. *Bioinformatics*, *24*(23), 2786–2787. https://doi.org/10.1093/bioinformatics/btn522

Hethcote, H. W. (2000). The Mathematics of Infectious Diseases. *SIAM Review*, *42*(4), 599–653. https://doi.org/10.1137/S0036144500371907

Hey, J., & Wakeley, J. (1997). A coalescent estimator of the population recombination rate. *Genetics*, *145*(3), 833–846. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9055092

Holland, J. J., Spindler, K., Horodyski, F., Grabau, E., Nichol, S., & VandePol, S. (1982). Rapid evolution of RNA genomes. *Science*, *215*(4540), 1577–1585. https://doi.org/10.1126/science.7041255

Holme, P., & Kim, B. J. (2002). Growing scale-free networks with tunable clustering. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, *65*(2), 2–5. https://doi.org/10.1103/PhysRevE.65.026107

Holmes, E. C. (2003). Patterns of intra- and interhost nonsynonymous variation reveal strong purifying selection in dengue virus. *Journal of Virology*, *77*(20), 11296–11298. https://doi.org/10.1128/JVI.77.20.11296-11298.2003

Holmes, E. C. (2010). The RNA Virus Quasispecies: Fact or Fiction? *Journal of Molecular Biology*, *400*(3), 271–273. https://doi.org/10.1016/j.jmb.2010.05.032

Holmes, E. C., & Moya, A. (2002a). Is the quasispecies concept relevant to RNA viruses? *Journal of Virology*, *76*(1), 460–465. https://doi.org/10.1128/JVI.76.1.460-462.2002

Holmes, E. C., & Moya, A. (2002b). Is the quasispecies qoncept relevant to RNA viruses? *Journal of Virology*, *76*(1), 460–462. https://doi.org/10.1128/JVI.76.1.460

Hudson, R. R., & Kaplan, N. L. (1988). The coalescent process in models with selection and recombination. *Genetics*, *120*(3), 831–840. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/3147214

Jenkins, G. M., Worobey, M., Woelk, C. H., & Holmes, E. C. (1998). Evidence for the Non-quasispecies Evolution of RNA Viruses. *Molecular Biology and Evolution*, *18*(6), 987–994. https://doi.org/10.1093/oxfordjournals.molbev.a003900

Jombart, T., Eggo, R. M., Dodd, P. J., & Balloux, F. (2011). Reconstructing disease outbreaks from genetic data: a graph approach. *Heredity*, *106*(2), 383–390. https://doi.org/10.1038/hdy.2010.78

Kaplan, N. L., Darden, T., & Hudson, R. R. (1988). The coalescent process in models with selection. *Genetics*, *120*(3), 819–829. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/3066685

Kawashima, K. D. (2017). Contagion. Retrieved from https://github.com/kentwait/contagion

Kawashima, K. D., Matsumoto, T., & Akashi, H. (2016). Disease Outbreaks: Critical Biological Factors and Control Strategies (pp. 173–204). Springer, Cham. https://doi.org/10.1007/978-3-319-39812-9_10

Keeling, M. J. (2005). The implications of network structure for epidemic dynamics. *Theoretical Population Biology*, *67*(1), 1–8. https://doi.org/10.1016/j.tpb.2004.08.002

Keeling, M. J., & Rohani, P. (2007). *Modeling Infectious Diseases in Humans and Animals. Public Health*. Princeton University Press. https://doi.org/10.1097/01.ede.0000254692.80550.60

Kennedy, D. A., & Dwyer, G. (2018). Effects of multiple sources of genetic drift on pathogen variation within hosts. *PLOS Biology*, *16*(3), e2004444.

https://doi.org/10.1371/journal.pbio.2004444

Kermack, W. O., & McKendrick, A. G. (1927). A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *115*(772), 700–721. https://doi.org/10.1098/rspa.1927.0118

Kingman, J. F. C. (1982). On the genealogy of large populations. *Journal of Applied Probability*, *19*(A), 27–43. https://doi.org/10.2307/3213548

Koelle, K., Cobey, S., Grenfell, B. T., & Pascual, M. (2006). Epochal evolution shapes the phylodynamics of interpandemic influenza A (H3N2) in humans. *Science (New York, N.Y.)*, *314*(5807), 1898–1903. https://doi.org/10.1126/science.1132745

Ladner, J. T. T., Wiley, M. R. R., Mate, S., Dudas, G., Prieto, K., Lovett, S., … Palacios, G. (2015). Evolution and Spread of Ebola Virus in Liberia, 2014–2015. *Cell Host & Microbe*, *18*(6), 659–669. Retrieved from https://www.sciencedirect.com/science/article/pii/S193131281500462X?via%3Dihub

Lalić, J., Cuevas, J. M., & Elena, S. F. (2011). Effect of host species on the distribution of mutational fitness effects for an RNA virus. *PLoS Genetics*, *7*(11), e1002378. https://doi.org/10.1371/journal.pgen.1002378

Lauring, A. S., Acevedo, A., Cooper, S. B., & Andino, R. (2012). Codon usage determines the mutational robustness, evolutionary capacity, and virulence of an RNA virus. *Cell Host and Microbe*, *12*(5), 623–632. https://doi.org/10.1016/j.chom.2012.10.008

Lauring, A. S., & Andino, R. (2010). Quasispecies Theory and the Behavior of RNA Viruses. *PLoS Pathogens*, *6*(7), e1001005. https://doi.org/10.1371/journal.ppat.1001005

Lázaro, E., Escarmís, C., Domingo, E., & Manrubia, S. C. (2002). Modeling viral genome fitness evolution associated with serial bottleneck events: evidence of stationary states of fitness. *Journal of Virology*, *76*(17), 8675–8681. https://doi.org/10.1128/JVI.76.17.8675-8681.2002

Leventhal, G. E., Hill, A. L., Nowak, M. A., & Bonhoeffer, S. (2015). Evolution and emergence of infectious diseases in theoretical and real-world networks. *Nature Communications*, *6*, 6101. https://doi.org/10.1038/ncomms7101

Levins, R. (1968). *Evolution in Changing Environments*. Princeton University Press.

Levins, R. (1969). Some Demographic and Genetic Consequences of Environmental Heterogeneity for Biological Control. *Bulletin of the Entomological Society of America*, *15*(3), 237–240. https://doi.org/10.1093/besa/15.3.237

Li, H., & Roossinck, M. J. (2004). Genetic Bottlenecks Reduce Population Variation in an Experimental RNA Virus Population Genetic Bottlenecks Reduce Population Variation in an Experimental RNA Virus Population, *78*(19). https://doi.org/10.1128/JVI.78.19.10582

Li, H., & Stephan, W. (2006). Inferring the Demographic History and Rate of Adaptive Substitution in Drosophila. *PLoS Genetics*, *2*(10), e166. https://doi.org/10.1371/journal.pgen.0020166

Lloyd-Smith, J. O., Funk, S., McLean, A. R., Riley, S., & Wood, J. L. N. (2015). Nine challenges in modelling the emergence of novel pathogens. *Epidemics*, *10*, 35–39. https://doi.org/10.1016/j.epidem.2014.09.002

Lloyd-Smith, J. O., Schreiber, S. J., Kopp, P. E., & Getz, W. M. (2005). Superspreading and the effect of individual variation on disease emergence. *Nature*, *438*(7066), 355–359. https://doi.org/10.1038/nature04153

Lloyd, A. L. (2001). Realistic Distributions of Infectious Periods in Epidemic Models: Changing Patterns of Persistence and Dynamics. *Theoretical Population Biology*, *60*(1), 59–71. https://doi.org/10.1006/TPBI.2001.1525

Lo, A. W., Tang, N. L., & To, K.-F. (2006). How the SARS coronavirus causes disease: host or organism? *The Journal of Pathology*, *208*(2), 142–151. https://doi.org/10.1002/path.1897

Long, J. C., Williams, R. C., & Urbanek, M. (1995). An E-M algorithm and testing strategy for multiple-locus haplotypes. *American Journal of Human Genetics*, *56*(3), 799–810. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/7887436

Lu, G., & Liu, D. (2012). SARS-like virus in the Middle East: A truly bat-related coronavirus causing human diseases. *Protein and Cell*, *3*(11), 803–805. https://doi.org/10.1007/s13238-012-2811-1

Luciani, F., & Alizon, S. (2009). The Evolutionary Dynamics of a Rapidly Mutating Virus within and between Hosts: The Case of Hepatitis C Virus. *PLoS Comput Biol*, *5*(11), e1000565. https://doi.org/10.1371/journal.pcbi.1000565

Maruyama, T. (1969). Genetic correlation in the stepping stone model with non-symmetrical migration rates. *J. Appl. Probab.*, *6*(3), 463–477.

Maruyama, T. (1970). Rate of decrease of genetic variability in a subdivided population. *Biometrika*, *57*(2), 299–311. https://doi.org/10.1093/biomet/57.2.299

Maruyama, T., & Yasuda, N. (1970). Use of Graph Theory in Computation of Inbreeding and Kinship Coefficients. *Biometrics*, *26*(2), 209–219. https://doi.org/10.2307/2529069

Maynard Smith, J., & Price, G. R. (1973). The logic of animal conflict. *Nature*, *246*(5427), 15–18. https://doi.org/10.1038/246015a0

McCrone, J. T., Woods, R. J., Martin, E. T., Malosh, R. E., Monto, A. S., & Lauring, A. S. (2018). Stochastic processes constrain the within and between host evolution of influenza virus. *ELife*, *7*, e35962. https://doi.org/10.7554/eLife.35962

McDonald, J. H., & Kreitman, M. (1991). Adaptive protein evolution at the Adh locus in Drosophila. *Nature*, *351*(6328), 652–654. https://doi.org/10.1038/351652a0

Messenger, S. L., Molineux, I. J., & Bull, J. J. (1999). Virulence evolution in a virus obeys a trade-off. *Proceedings of the Royal Society B: Biological Sciences*, *266*(1417), 397–404. https://doi.org/10.1098/rspb.1999.0651

Metz, J. A. J., Nisbet, R. M., & Geritz, S. A. H. (1992). How should we define ‘fitness’ for general ecological scenarios? *Trends in Ecology & Evolution*, *7*(6), 198–202. https://doi.org/10.1016/0169-5347(92)90073-K

Meyers, L. A., Newman, M. E. J., Martin, M., & Schrag, S. J. (2003). Applying network theory to epidemics: Control measures for Mycoplasma pneumoniae outbreaks. *Emerging Infectious Diseases*, *9*(2), 204–210. https://doi.org/10.3201/eid0902.020188

Meyers, L. A., Pourbohloul, B., Newman, M. E. J., Skowronski, D. M., & Brunham, R. C. (2005). Network theory and SARS: predicting outbreak diversity. *Journal of Theoretical Biology*, *232*(1), 71–81. https://doi.org/10.1016/j.jtbi.2004.07.026

Moore, C., & Newman, M. E. J. (2000). Epidemics and percolation in small-world networks. *Physical Review. E, Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, *61*(5 Pt B), 5678–5682. https://doi.org/10.1103/PhysRevE.61.5678

Muller, H. J. (1964). The relation of recombination to mutational advance. *Mutation Research - Fundamental and Molecular Mechanisms of Mutagenesis*, *1*(1), 2–9. https://doi.org/10.1016/0027-5107(64)90047-8

Murcia, P. R., Baillie, G. J., Daly, J. M., Elton, D., Jervis, C., Mumford, J. A., … Wood, J. L. N. (2010). Intra- and interhost evolutionary dynamics of equine influenza virus. *Journal of Virology*, *84*(14), 6943–6954. https://doi.org/10.1128/JVI.00112-10

Murillo, L. N., Murillo, M. S., & Perelson, A. S. (2013). Towards multiscale modeling of influenza infection. *Journal of Theoretical Biology*, *332*, 267–290. https://doi.org/10.1016/j.jtbi.2013.03.024

Mustonen, V., & Lässig, M. (2009). From fitness landscapes to seascapes: non-equilibrium dynamics of selection and adaptation. *Trends in Genetics*, *25*(3), 111–119. https://doi.org/10.1016/J.TIG.2009.01.002

Nagylaki, T. (1982). Geographical invariance in population genetics. *Journal of Theoretical Biology*, *99*(1), 159–172.

Nagylaki, T. (1985). Homozygosity, Effective Number of Alleles, and Interdeme Differentiation in Subdivided Populations. *Proceedings of the National Academy of Sciences of the United States of America*, *82*(24), 8611–8613.

Nagylaki, T. (1998). Fixation indices in subdivided populations. *Genetics*, *148*(3), 1325–1332.

Nasser, W., Beres, S. B., Olsen, R. J., Dean, M. A., Rice, K. A., Long, S. W., … Musser, J. M. (2014). Evolutionary pathway to increased virulence and epidemic group A Streptococcus disease derived from 3,615 genome sequences. *Proceedings of the National Academy of Sciences*, *111*(17), E1768–E1776. https://doi.org/10.1073/pnas.1403138111

Nei, M. (1973). Analysis of Gene Diversity in Subdivided Populations. *Proc. Nat. Acad. Sci. USA*, *70*(12), 3321–3323. https://doi.org/10.1073/pnas.70.12.3321

Neuhauser, C., & Krone, S. M. (1997). The genealogy of samples in models with selection. *Genetics*, *145*(2), 519–534. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9071604

Newman, M. E. J. (2002). The spread of epidemic disease on networks, 12. Statistical Mechanics; Disordered Systems and Neural Networks; Quantitative Biology. https://doi.org/10.1103/PhysRevE.66.016128

Newman, M. E. J. (2010). *Networks*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

Nielsen, R., Hubisz, M. J., Hellmann, I., Torgerson, D., Andrés, A. M., Albrechtsen, A., … Clark, A. G. (2009). Darwinian and demographic forces affecting human protein coding genes. *Genome Research*, *19*(5), 838–849. https://doi.org/10.1101/gr.088336.108

Notohara, M. (1990). The coalescent and the genealogical process in geographically structured population. *Journal of Mathematical Biology*, *29*(1), 59–75. https://doi.org/10.1007/BF00173909

Novella, I. S., Presloid, J. B., Smith, S. D., & Wilke, C. O. (2011). Specific and nonspecific host adaptation during arboviral experimental evolution. *Journal of Molecular Microbiology and Biotechnology*, *21*(1–2), 71–81. https://doi.org/10.1159/000332752

Otto, S. P., & Whitlock, M. C. (1997). The probability of fixation in populations of changing size. *Genetics*, *146*(2), 723–733. https://doi.org/10.1534/genetics.104.040089

Pannell, J. R., & Charlesworth, B. (2000). Effects of metapopulation processes on measures of genetic diversity. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *355*(1404), 1851–1864. https://doi.org/10.1098/rstb.2000.0740

Papaïx, J., Burdon, J. J., Lannou, C., & Thrall, P. H. (2014). Evolution of pathogen specialisation in a host metapopulation: joint effects of host and pathogen dispersal. *PLoS Computational Biology*, *10*(5), e1003633. https://doi.org/10.1371/journal.pcbi.1003633

Park, M., Loverdo, C., Schreiber, S. J., & Lloyd-Smith, J. O. (2013). Multiple scales of selection influence the evolutionary emergence of novel pathogens. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *368*(1614), 20120333. https://doi.org/10.1098/rstb.2012.0333

Pepin, K. M., Volkov, I., Banavar, J. R., Wilke, C. O., & Grenfell, B. T. (2010). Phenotypic differences in viral immune escape explained by linking within-host dynamics to host-population immunity. *Journal of Theoretical Biology*, *265*(4), 501–510. https://doi.org/10.1016/j.jtbi.2010.05.036

Pollak, E. (1966). On the Survival of a Gene in a Subdivided Population. *Journal of Applied Probability*, *3*(1), 142–155.

Poon, L. L. M., Song, T., Rosenfeld, R., Lin, X., Rogers, M. B., Zhou, B., … Ghedin, E. (2016). Quantifying influenza virus diversity and transmission in humans. *Nature Genetics*, *48*(2), 195–200. https://doi.org/10.1038/ng.3479

Pybus, O. G., Fraser, C., & Rambaut, A. (2013). Evolutionary epidemiology: preparing for an age of genomic plenty. *Philosophical Transactions of the Royal Society of London.*

Series B, Biological Sciences, *368*(1614), 20120193. https://doi.org/10.1098/rstb.2012.0193

Querec, T. D., Akondy, R. S., Lee, E. K., Cao, W., Nakaya, H. I., Teuwen, D., … Pulendran, B. (2009). Systems biology approach predicts immunogenicity of the yellow fever vaccine in humans. *Nature Immunology*, *10*(1), 116–125. https://doi.org/10.1038/ni.1688

Read, J. M., & Keeling, M. J. (2006). Disease evolution across a range of spatio-temporal scales. *Theoretical Population Biology*, *70*(2), 201–213. https://doi.org/10.1016/j.tpb.2006.04.006

Reid, A. H., Fanning, T. G., Hultin, J. V, & Taubenberger, J. K. (1999). Origin and evolution of the 1918 &quot;Spanish&quot; influenza virus hemagglutinin gene. *Proceedings of the National Academy of Sciences of the United States of America*, *96*(4), 1651–1656. https://doi.org/10.1073/PNAS.96.4.1651

Rico-hesse, R. (2003). Microevolution and virulence of dengue viruses. *Adv Virus Res*, *59*, 315–341. https://doi.org/10.1016/S0065-3527(03)59009-1

Ruiz-Jarabo, C. M., Arias, A., Baranowski, E., Escarmís, C., & Domingo, E. (2000). Memory in Viral Quasispecies. *Journal of Virology*, *74*(8), 3543–3547. https://doi.org/10.1128/JVI.74.8.3543-3547.2000

Ruiz-Jarabo, C. M., Miller, E., Gómez-Mariano, G., & Domingo, E. (2003). Synchronous loss of quasispecies memory in parallel viral lineages: A deterministic feature of viral quasispecies. *Journal of Molecular Biology*, *333*(3), 553–563. https://doi.org/10.1016/j.jmb.2003.08.054

Salathe, M., & Jones, J. H. (2010). Dynamics and Control of Diseases in Networks with Community Structure. *PLoS Computational Biology*, *6*(4), e1000736. https://doi.org/10.1371/journal.pcbi.1000736

Sanford, J., Baumgardner, J., Brewer, W., Gibson, P., & Remine, W. (2007). Mendel's Accountant: a biologically realistic forward-time population genetics program. *Scalable Computing: Practice and Experience*, *8*(2), 147–165.

Sanjuán, R., Cuevas, J. M., Furió, V., Holmes, E. C., & Moya, A. (2007). Selection for robustness in mutagenized RNA viruses. *PLoS Genetics*, *3*(6), 0939–0946. https://doi.org/10.1371/journal.pgen.0030093

Sanjuán, R., & Domingo-Calap, P. (2016). Mechanisms of viral mutation. *Cellular and Molecular Life Sciences : CMLS*, *73*(23), 4433–4448. https://doi.org/10.1007/s00018-016-2299-6

Sawyer, S. A., & Hartl, D. L. (1992). Population genetics of polymorphism and divergence. *Genetics*, *132*(4).

Schreiber, M. J., Holmes, E. C., Ong, S. H., Soh, H. S. H., Liu, W., Tanner, L., … Hibberd, M. L. (2009). Genomic epidemiology of a dengue virus epidemic in urban Singapore. *Journal of Virology*, *83*(9), 4163–4173. https://doi.org/10.1128/JVI.02445-08

Seabloom, E. W., Borer, E. T., Gross, K., Kendig, A. E., Lacroix, C., Mitchell, C. E., … Power, A. G. (2015). The community ecology of pathogens: coinfection, coexistence and community composition. *Ecology Letters*, *18*(4), 401–415. https://doi.org/10.1111/ele.12418

Sierra, S., Dávila, M., Lowenstein, P. R., & Domingo, E. (2000). Response of foot-and-mouth disease virus to increased mutagenesis: influence of viral load and fitness in loss of infectivity. *Journal of Virology*, *74*(18), 8316–8323. https://doi.org/10.1128/JVI.74.18.8316-8323.2000

Slatkin, M. (1977). Gene flow and genetic drift in a species subject to frequent local extinctions. *Theoretical Population Biology*, *12*(3), 253–262.

Slatkin, M. (1981). Fixation probabilities and fixation times in a subdivided population. *Evolution*, *35*(3), 477–488. https://doi.org/10.2307/2408196

Smith, D. J., Lapedes, A. S., de Jong, J. C., Bestebroer, T. M., Rimmelzwaan, G. F., Osterhaus, A. D. M. E., & Fouchier, R. A. M. (2004). Mapping the antigenic and genetic evolution of influenza virus. *Science (New York, N.Y.)*, *305*(5682), 371–376. https://doi.org/10.1126/science.1097211

Smith, G. J. D., Vijaykrishna, D., Bahl, J., Lycett, S. J., Worobey, M., Pybus, O. G., … Rambaut, A. (2009). Origins and evolutionary genomics of the 2009 swine-origin H1N1 influenza A epidemic. *Nature*, *459*(7250), 1122–1125. https://doi.org/10.1038/nature08182

Starnini, M., Machens, A., Cattuto, C., Barrat, A., & Pastor-Satorras, R. (2013). Immunization strategies for epidemic processes in time-varying contact networks. *Journal of Theoretical Biology*, *337*, 89–100. https://doi.org/10.1016/j.jtbi.2013.07.004

Stern, A., Bianco, S., Yeh, M. Te, Wright, C., Butcher, K., Tang, C., … Andino, R. (2014). Costs and benefits of mutational robustness in RNA viruses. *Cell Reports*, *8*(4), 1026–1036. https://doi.org/10.1016/j.celrep.2014.07.011

Stolerman, L., Coombs, D., & Boatto, S. (2015). SIR-Network Model and Its Application to Dengue Fever. *SIAM Journal on Applied Mathematics*, *75*(6), 2581–2609. https://doi.org/10.1137/140996148

Strimmer, K., & Pybus, O. G. (2001). Exploring the Demographic History of DNA Sequences Using the Generalized Skyline Plot. *Molecular Biology and Evolution*, *18*(12), 2298–2305. https://doi.org/10.1093/oxfordjournals.molbev.a003776

Susi, H., Barrès, B., Vale, P. F., & Laine, A.-L. (2015). Co-infection alters population dynamics of infectious disease. *Nature Communications*, *6*(1), 5975. https://doi.org/10.1038/ncomms6975

Tajima, F. (1989). The effect of change in population size on DNA polymorphism. *Genetics*, *123*(3), 597–601. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/2599369

Thrall, P. H., & Burdon, J. (1997). Host-pathogen dynamics in a metapopulation context: The ecological and evolutionary consequences of being spatial. *Journal of Ecology*, *85*(6), 743–753. https://doi.org/10.2307/2960598

Thrall, P. H., & Burdon, J. J. (2003). Evolution of virulence in a plant host-pathogen metapopulation. *Science*, *299*(5613), 1735–1737. https://doi.org/10.1126/science.1080070

Turner, P. E., & Elena, S. F. (2000). Cost of host radiation in an RNA virus. *Genetics*, *156*(4), 1465–1470. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1461356&tool=pmcentrez&rendertype=abstract

Vasilakis, N., Deardorff, E. R., Kenney, J. L., Rossi, S. L., Hanley, K. A., & Weaver, S. C. (2009). Mosquitoes Put the Brake on Arbovirus Evolution: Experimental Evolution Reveals Slower Mutation Accumulation in Mosquito Than Vertebrate Cells. *PLoS Pathogens*, *5*(6), e1000467. https://doi.org/10.1371/journal.ppat.1000467

Vergu, E., Busson, H., & Ezanno, P. (2010). Impact of the Infection Period Distribution on the Epidemic Spread in a Metapopulation Model. *PLoS ONE*, *5*(2), e9371. https://doi.org/10.1371/journal.pone.0009371

Vignuzzi, M., Stone, J. K., Arnold, J. J., Cameron, C. E., & Andino, R. (2006). Quasispecies diversity determines pathogenesis through cooperative interactions in a viral population. *Nature*, *439*(7074), 344–348. https://doi.org/10.1038/nature04388

Volz, E. M., Kosakovsky Pond, S. L., Ward, M. J., Leigh Brown, A. J., & Frost, S. D. W. (2009). Phylodynamics of Infectious Disease Epidemics. *Genetics*, *183*(4), 1421–1430. https://doi.org/10.1534/genetics.109.106021

Vuilleumier, S., Yearsley, J. M., & Perrin, N. (2008). The fixation of locally beneficial alleles in a metapopulation. *Genetics*, *178*(1), 467–475. https://doi.org/10.1534/genetics.107.081166

Wakeley, J. (1998). Segregating sites in Wright's island model. *Theoretical Population Biology*, *53*(2), 166–174. https://doi.org/10.1006/tpbi.1997.1355

Wakeley, J. (2003). Polymorphism and divergence for Island-Model species. *Genetics*, *163*(1), 411–420.

Wakeley, J. (2005, January). The limits of theoretical population genetics. *Genetics*. Genetics Society of America. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/15677744

Whitlock, M. C., & Barton, N. H. (1997). The effective size of a subdivided population. *Genetics*, *146*(1), 427–441. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9136031

Wilke, C. O. (2005). Quasispecies theory in the context of population genetics. *BMC Evolutionary Biology*, *5*(1), 44. https://doi.org/10.1186/1471-2148-5-44

Wilke, C. O., Wang, J. L., Ofria, C., Lenski, R. E., & Adami, C. (2001). Evolution of digital organisms at high mutation rates leads to survival of the flattest. *Nature*, *412*(6844), 331–333. https://doi.org/10.1038/35085569

Woolhouse, M. E. J., & Gowtage-Sequeria, S. (2005). Host range and emerging and reemerging pathogens. *Emerging Infectious Diseases*, *11*(12), 1842–1847.

https://doi.org/10.3201/eid1112.050997

Wright, S. (1931). Evolution in Mendelian Populations. *Genetics*, *16*(2), 97–159. https://doi.org/10.1007/BF02459575

Wright, S. (1943). Isolation by Distance. *Genetics*, *28*(2), 114–138. https://doi.org/Article

Wright, S. (1949). The genetical structure of populations. *Annals of Eugenics*, *15*(1), 323–354. https://doi.org/10.1111/j.1469-1809.1949.tb02451.x

Xie, X. (2011). The Site-Frequency Spectrum of Linked Sites. *Bulletin of Mathematical Biology*, *73*(3), 459–494. https://doi.org/10.1007/s11538-010-9534-3

Xue, K. S., Stevens-Ayers, T., Campbell, A. P., Englund, J. A., Pergam, S. A., Boeckh, M., & Bloom, J. D. (2017). Parallel evolution of influenza across multiple spatiotemporal scales. *ELife*, *6*, e26875. https://doi.org/10.7554/eLife.26875

Yu, I. T., Xie, Z. H., Tsoi, K. K., Chiu, Y. L., Lok, S. W., Tang, X. P., … Sung, J. J. Y. (2007). Why Did Outbreaks of Severe Acute Respiratory Syndrome Occur in Some Hospital Wards but Not in Others? *Clinical Infectious Diseases*, *44*(8), 1017–1025. https://doi.org/10.1086/512819

Zeng, K. (2010). A Simple Multiallele Model and Its Application to Identifying Preferred-Unpreferred Codons Using Polymorphism Data. *Molecular Biology and Evolution*, *27*(6), 1327–1337. https://doi.org/10.1093/molbev/msq023

Zhang, C.-Y., Wei, J.-F., & He, S.-H. (2006). Adaptive evolution of the spike gene of SARS coronavirus: changes in positively selected sites in different epidemic groups. *BMC Microbiology*, *6*, 88. https://doi.org/10.1186/1471-2180-6-88

Zwart, M. P., & Elena, S. F. (2014). Testing the Independent Action Hypothesis of Plant Pathogen Mode of Action: A Simple and Powerful New Approach. *Phytopathology*, 1–26. https://doi.org/10.1094/PHYTO-04-14-0111-R

Zwart, M. P., Hemerik, L., Cory, J. S., de Visser, J. A. G. M., Bianchi, F. J. J. A., Van Oers, M. M., … Van der Werf, W. (2009). An experimental test of the independent action hypothesis in virus-insect pathosystems. *Proceedings. Biological Sciences*, *276*(1665), 2233–2242. https://doi.org/10.1098/rspb.2009.0064

APPENDIX I
CONTAGION CONFIGURATION FORMAT

INTRODUCTION

The configuration file is the main method to set-up and control the behavior of a simulation. The configuration file follows the TOML specification and is divided into different sections grouped by the role of the parameters.

The TOML standard follows certain rules on how to declare and name sections and variables. For instance, a title enclosed by square brackets `[]` marks the start of a new section and all parameters declared after become part of that section until a new section title is encountered. To assign values to parameters, simply use an equal sign = such that `parameter_name = value`. If the value is a string, enclose the string in quotation marks `""`. If the parameter accepts a list of values, enclose the list of values in square brackets `[]` and separate each value using a comma `,` .

There are two types of parameters in Contagion - global and local parameters. Global parameters affect the entire simulation, such as the size and the duration of the simulation, and how the data generated from the simulation is logged and stored. Global parameters can only be declared once and have only one constant value for the entire simulation. Parameters in the `simulation` and the `logging` sections of the configuration file are global parameters.

Parameters that would only affect a part of the simulation are called local parameters. For example, you want to simulate the evolution of pathogens across two different host species where viral mutation rate is higher in one compared to the other. Under this scenario, the population of hosts in the simulation need to be divided into two groups and each group requires a distinct viral mutation rate parameter. Thus, mutation rate is a local parameter that can be configured to have multiple values, each affecting only a particular subset of the population. Parameters in the `intrahost_model`, `fitness_model`, `transmission_model`, and `stop_condition` sections are local parameters. Note that while local parameters can be set to affect only a portion of the simulation, these parameters can also be used to assign a global behavior to the simulation.

GLOBAL PARAMETERS

Simulation parameters

The `simulation` section sets the type of epidemic model to run, size and shape of the host population network, the number of pathogen genetic sites to model, and the number of independent realizes to run.

Figure 27. Simulation parameters section format

```
[simulation]

epidemic_model = "sir"
coinfection = false

host_popsize = 10
host_network_path = "network.txt"

num_sites = 10000
expected_characters = ["A", "T", "C", "G"]
pathogen_path = "pathogens.fa"

num_instances = 1
```

To mark the start of the simulation section, the section title is enclosed by square brackets `[]` and all parameters declared after become part of the section until a new section title is encountered. This behavior is not only limited to the Contagion configuration file but is part of the TOML specification.

epidemic_model parameter

The `epidemic_model` parameter specifies the type of epidemic model to simulate. Contagion uses the concept of a compartmental model to determine the status of each host individual in the simulation. During the simulation, a host can have only one status (belong to only one compartment) at every time step. The initial (default) state is the susceptible state. This indicates that the host is not currently infected. When a pathogen infects a host, the host moves from susceptible and enters the infected state. The duration of the infection can be determined using a transition probability, a set length of time, or depend on the fitness of infecting pathogens within the host. In some models, infection is chronic and persists throughout the simulation time, whereas others have acute infection.

Table 5. Epidemic models implemented in Contagion.

Keyword	Epidemic model	Description
si	susceptible-infected	Hosts infected with the pathogen remain infected until the end of the simulation
sis	susceptible-infected-susceptible	Hosts recover from the infection and immediately become susceptible again. After infection, all pathogens within the host is removed. However, recovered hosts can be reinfected.
sir	susceptible-infected-removed	Hosts recover from the infection. After infection, all pathogens within the host is removed. Hosts cannot be reinfected after recovery.
sirs	susceptible-infected-removed-susceptible	Similar to \`sir\` except that hosts can become susceptible again and may be reinfected.
sei	susceptible-exposed-infectious	Hosts infected by the pathogen do not immediately become infectious. This model introduces a lag time between infection and the ability to spread the infection.
seir	susceptible-exposed-infected-removed	Similar to \`sir\` except a lag time exists between infection and the ability to spread the disease.
seirs	susceptible-exposed-infected-removed-susceptible	Similar to \`sirs\` except a lag time exists between infection and the ability to spread the disease.
endtrans	susceptible-infected-removed	Based on \`sir\`, this model conditions transmission of the infection at the last day of infection.

coinfection parameter

The `coinfection` parameter indicates whether mixed infections are allowed. Originally, pathogens can only infect hosts that are in the susceptible state. However, this behavior makes infected individuals temporarily immune to inoculation while in the infected state. To address this peculiar dynamic, coinfection allows pathogens to both infect susceptible hosts and currently infected individuals.

Table 6. List of coinfection parameter values

Value	Description
true	Allows pathogens to infect susceptible and infected individuals.
false	Pathogens can only be spread to susceptible individuals.

Note: Text values are care sensitive.

host_popsize parameter

The `host_popsize` parameter describes the number of hosts to be modeled in the simulation. The value of this parameter must be an integer and should match the number of hosts in the host network configuration file.

host_network_path parameter

The `host_network_path` indicates the location of the host network configuration file. This path can be either an absolute path or the relative path. However, it is recommended to use the absolute path for clarity.

Figure 28. Host network text file format

```
# Ignores lines that begin with a hash
1   2
2   1
1   3
3   1
1   4
```

```
4   1
```

Each line in the host network configuration file indicates a one-way connection between two hosts by ID. To specify an undirected connection, declare the host pair twice in the forward and reverse directions. Individual transmission probabilities can also be specified for each connection by adding a third value.

Figure 29. Host network text file extended format

```
# Ignores lines that begin with a hash
1   2   0.5
2   1   0.7
1   3   0.1
3   1   0.9
1   4   0.8
4   1   0.8
```

Note: Use absolute paths to indicate the location of the host network configuration file.

Currently, Contagion uses a static host network and does not use networks that can change over time.

num_sites parameter

The `num_sites` parameter indicates the number of genetic sites to model during the simulation. The value must be an integer greater than 0.

In Contagion, a site is the smallest evolvable unit of information. Each site holds a categorical state that can be mutated based on a given transition matrix of probabilities. The sequence of sites and their states make up the simulated genome of the pathogen.

The number of sites depends on the simulation objective. If the simulation intends to model the molecular evolution at the nucleotide level, then each site should represent each nucleotide in the sequence. On the other hand, if the simulation is concerned about codon or protein evolution, then each site should be modeled after a codon or an amino acid respectively. Another way to model evolution is to consider a site as a locus that holds an allele. Under this view, one site may be sufficient to model the dynamics of the system.

pathogen_path parameter

The `pathogen_path` parameter indicates the location of the pathogen sequence file. This path can be either an absolute path or the relative path. However, it is recommended to use the absolute path for clarity.

Figure 30. Pathogen sequence file format.

```
# Ignores lines that begin with a hash
% U:0 P:1
>pathogen1 h:0
UUPUUPUPUUPPUUPUUPUPPPUU
>pathogen2 h:0
PUPUUPUPUPUPPUUUUPPPPPPP
```

The pathogen sequence file is based on the FASTA file format with some additional formatting rules. The pathogen sequence file must have a line starting with a % character before any sequence. This line lists the expected characters and its numerical conversion. Then, ID and sequence lines follow using the standard FASTA formatting guidelines. Lines that begin with # are ignored.

One exception to the FASTA format that Contagion still accepts is the case of non-unique identifiers. In Figure 31, the sequence IDs are not unique and is not considered valid FASTA formatting, but is a valid pathogen sequence file.

Figure 31. Pathogen sequence file format with non-unique identifiers.

```
# Ignores lines that begin with a hash
% U:0 P:1
>h:0
UUPUUPUPUUPPUUPUUPUPPPUU
>h:0
```

```
PUPUUPUPUPUPPUUUUPPPPPPP
```

Note: Use absolute paths to indicate the location of the pathogen sequence file.

num_instances parameter

The `num_instances` parameter specifies the number of independent realizations of the simulation to perform. This is similar to the concept of replication. If `num_instances` is greater than 1, then the simulation will be repeated performed using the same set of input parameters. Since the processes in the simulation are stochastic, then each realized simulation may not necessarily reproduce the results from previous trials.

Logging parameters

Figure 32. Logging parameters section format

```
[logging]

log_freq = 1
log_path = "examples/run.log"
```

log_freq parameter

The `log_freq` parameter indicates how often data generated by the simulation will be saved to disk. Setting `log_freq` to 1 will save all the data generated for every step in the simulation. On the other hand, setting it to a value greater than 1 means that data will be saved at periodic intervals only.

log_path parameter

The `log_path` parameter indicates where to save the data generated during the simulation. This path can be either an absolute path or the relative path. However, it is recommended to use the absolute path for clarity.

Note: Use absolute paths to indicate the location of the pathogen sequence file.

HOST PARAMETERS

Contagion host-associated parameters are local parameters to facilitate parameter value heterogeneities in the host population. These parameters are divided into three sections - `intrahost_model`, `fitness_model`, and `transmission_model`. To associate each set of parameters, each section has a `host_ids` parameter to specify which hosts in the simulation adopt these parameter values. Thus, it is possible to make various configurations without having to redeclare parameters with shared values.

Consider a simulation with three groups of hosts such that selection on infecting pathogens acts differently among the three. This means that all groups share the same `intrahost_model` and `transmission_model` parameters and differ only in parameters found in the `fitness_model` section. To simplify the set-up, Contagion allows you to declare the `intrahost_model` and `transmission_model` sections once and associate it to all hosts in the simulation using the `host_ids` parameter. These two models are declared only once because every host, regardless of the host group it belongs to share the same parameter configurations. On the other hand, the `fitness_model` section should be different among the three groups to reflect their different properties. To specify the differences in fitness, the `fitness_model` section can be declared three times, once for each host group and associated to that group by listing the member host IDs in the `host_ids` parameter.

Unlike the `simulation` and `logging` sections, `intrahost_model`, `fitness_model`, and `transmission_model` (and also `stop_condition`) can be declared more than once in the configuration. To reflect this difference, the section titles are enclosed in double square brackets `[[]]` instead of single square brackets `[]`.

Intrahost model

The intrahost model specifies the parameters that affect the behavior of the pathogen within the host it is infecting. Parameters associated with the intrahost model describe how to model processes of replication and recombination.

Figure 33. Intrahost model section format.

```
[[intrahost_model]]

model_name = "no_mutation_no_selection"
host_ids = [0,1,2,3,4,5,6,7,8,9]

mutation_rate = 0.0
transition_matrix = [
  [ 0.0e+00, 0.0e+00 ],
  [ 0.0e+00, 0.0e+00 ],
]

recombination_rate = 0.0

replication_model = "constant"
max_pop_size = 1000000

infected_duration = 10
```

model_name parameter

The `model_name` parameter can be used to set a descriptive name to label the given intrahost model.

host_ids parameter

The `host_ids` parameter specifies which hosts in the simulation will be associated to this particular intrahost model. To specify the hosts, the `host_ids` parameter takes on a list of one or more host IDs.

mutation_rate parameter

The `mutation_rate` parameter defines the genetic mutation rate of the pathogen during replication within the host. The value of the `mutation_rate` parameter refers to the average mutation rate for each site and for each generation.

transition_matrix parameter

The `transition_matrix` parameter specifies the transition rate matrix that the program will use to determine the identities of the new mutations.

To define a transition matrix, the program expects a nested list of lists where (1) each inner list is a row in the matrix, and (2) the number of elements of the inner list is equal to the number of inner lists such that it creates a square matrix.

Figure 34. Transition rate matrix for two characters (alleles) as a nested array.

```
transition_matrix = [ [ 0.0e+00, 1.0e-05 ], [ 1.0e-05, 0.0e+00 ] ]
```

Figure 35. Transition rate matrix for two characters (alleles) reformatted as a square matrix.

```
transition_matrix = [
  [ 0.0e+00, 1.0e-05 ],
  [ 1.0e-05, 0.0e+00 ],
]
```

For example, you are modeling a sequence where each site can take on one of two states. This means that the transition matrix for this scheme should have two inner lists with each list having two values.

The values within the inner lists must be floating-point numbers that contain a decimal point (.) among the digits. Without the decimal, whole numbers are automatically interpreted as integers by the program and will likely result in an input error. To declare a whole number as a floating-point type, a decimal must be appended at the end (for example `1.` or `1.0`).

Values can be specified using the normal decimal notation (for example 1000.0) or by scientific notation (`1.0e3`). For values greater than the ones place value, the exponent of the value in the scientific notation can be written as signed (`1.0e+3`) or unsigned integers (`1.0e3`). On the other hand, for values less than one (0.001), the exponent is always negative, and the sign must be included (`1.0e-3`).

Table 7. Proper floating-point value formatting.

Number	Valid	Description
0	False	Decimal point is missing. Number will be interpreted as an integer.
10	False	Decimal point is missing. Number will be interpreted as an integer.
0.	True	Trailing decimal point makes this number a floating-point type.
1.	True	Trailing decimal point makes this number a floating-point type.
0.0	True	Decimal point before the trailing zero makes this number a floating-point type.
10.2	True	Decimal point before the digits makes this number a floating-point type.
0.1	True	Decimal point after the leading zero makes this number a floating-point type.
.0	True	Leading decimal point also makes this number a floating-point type.
.12	True	Leading decimal point also makes this number a floating-point type.
1e1	False	Written in scientific notation but decimal point is missing. Number will be interpreted as an integer.
1.0e2	True	Written in scientific notation with a decimal point present.

Note: Transition matrix values must have a decimal point to be correctly interpreted. Contagion will return an error and not proceed if all values in the specified transition matrix are not floating-point values.

recombination_rate parameter

The `recombination_rate` parameter specifies the recombination rate between pathogen genomes that present within the host. The value of the `recombination_rate` parameter refers to the average number of recombination events per genome.

replication_model parameter

The `replication_model` parameter sets the demographics of the pathogen population within the host.

Table 8. Available replication models.

Keyword	Constant size	Description
constant	True	Constant population size within the host.
bh	False	Population size changes based on the Beverton-Holt population model. At high growth rates, the population size may suffer from periodic oscillations and may crash due to large fluctuations. If the `max_pop_size` parameter is set, the `max_pop_size` becomes the threshold population size such that the population size stays constant upon reaching the threshold number.
fitness	False	Population size changes based on the growth rates computed from the sequence of the pathogens.

max_pop_size parameter

The `max_pop_size` parameter is an optional parameter that refers to the threshold population size or the population size under the constant replication model. If the replication model is set to "bh" and the `max_pop_size` parameter is set, a threshold population size is overlaid and takes precedence over the Beverton-Holt population model.

infected_duration parameter

The `infected_duration` parameter specifies the average length of an infection.

Fitness model

The fitness model determines the way the pathogen's fitness value is calculated based on its genotype. Contagion currently implements three kinds of fitness models, namely, multiplicative fitness matrix, additive fitness matrix, and motif-based fitness. The first two fitness models use a site-fitness matrix to determine the fitness value contributed by each site based on its identity. On the other hand, the motif model also assigns fitness values based on the genotype, but it can consider the identity of more than one site to determine the corresponding fitness. The motif model is meant to simulate the effect of epistasis.

Figure 36. Fitness model section format.

```
[[fitness_model]]

model_name = "additive"
host_ids = [
 0
]
fitness_model = "additive"
fitness_model_path = "examples/fitpop/fm.txt"
```

model_name parameter

The `model_name` parameter can be used to set a descriptive name to label the given fitness model.

host_ids parameter

The `host_ids` parameter specifies which hosts in the simulation will be associated to this particular fitness model. To specify the hosts, the `host_ids` parameter takes on a list of one or more host IDs.

fitness_model parameter

The `fitness_model` parameter specifies the type of fitness model to determine the fitness of pathogens. Contagion implements two kinds of fitness models that uses either a fitness matrix or a list of motifs.

In the fitness matrix approach, each character at each site has an assigned fitness value. There are two fitness models to choose from - multiplicative and additive. The main difference lies in how the overall fitness of the pathogen is calculated. In a multiplicative fitness matrix, the overall fitness of the pathogen is calculated by getting the product of fitness contributions of all sites. Under this approach, the fitness contribution of one site does not cancel out the contribution of another site. On the other hand, in an additive fitness matrix, the overall fitness is calculated by taking the sum of the fitness contributions. This means that if one site has a positive fitness effect and another site has the same magnitude, but negative fitness effect, then these cancel each other out when computing the overall fitness of the pathogen.

Table 9. List of available fitness models

Keyword	Description
multiplicative	Multiplicative fitness matrix model. Fitness values are between 0 and 1 inclusive. This model takes the product of each site fitness value to determine the overall fitness of the pathogen. By taking the product, this model assumes that each site is independent of every other site and epistatic effects do not exist.
additive	Additive fitness matrix model. Fitness values can be any positive or negative rational number and zero. This model takes the sum of each site fitness value to get the overall fitness of the pathogen. Because of this, the positive fitness effect in one site can be canceled out by another site with negative fitness elsewhere.
additive_motif	(Additive) motif model. When a particular motif is present in the pathogen's sequence, the corresponding fitness value is assigned. When two or more motifs are involved, this model takes the sum of each motif's designated fitness value.

Note: Text values are care sensitive.

The other approach is called the motif model. In this case, overall fitness of a pathogen depends on whether one or motifs specified in the model are present in the sequence. If more than one motif is present, then the sum of the contributions is taken to determine the overall fitness. If no motif is found, the pathogen takes on some default value.

fitness_model_path parameter

The `fitness_model_path` parameter tells Contagion where to find the text file that contains the fitness matrix or list of motifs to use. Note that the fitness model type specified in the configuration file must match the contents of the file specified in the path. If a multiplicative fitness matrix model was specified but the file describes a motif model, Contagion will return an invalid input error.

Figure 37. Fitness model file format.

```
# Any line with the # as the first character is ignored.
default->0.0000 0.0000 0.0000 0.0000
0:  0.0000  0.0000  0.0000 -0.0003
10: 0.0000  0.0000 -0.0100  0.0000
24: 0.0000 -0.0003  0.0000  0.0000
35: 0.0000  0.0000  0.0000  0.0000
40: 0.0000  0.0000 -0.0100 -0.0003
49: 0.0000  0.0000  0.0000  0.0000
```

For multiplicative matrices, values should be in log-space, while for additive and motif models, values are expected to be in standard decimal format.

Transmission model

model_name parameter

The `model_name` parameter can be used to set a descriptive name to label the given fitness model.

host_ids parameter

The `host_ids` parameter specifies which hosts in the simulation will be associated to this particular fitness model. To specify the hosts, the `host_ids` parameter takes on a list of one or more host IDs.

transmission_prob parameter

The `transmission_prob` parameter sets the probability that a successful transmission event occurs for connection in an undirected network, or each outgoing connection in a directed network.

transmission_mode parameter

The `transmission_mode` parameter specifies whether the number of transmitted pathogens is constant or stochastic. There are two transmission modes currently implemented in Contagion - "constant" and "Poisson".

Table 10. List of available transmission modes

Keyword	Description
constant	A set number of pathogens are transmitted everytime a transmission event occurs.
poisson	The specified number of pathogens to be transmitted is the average number of the Poisson distribution. Everytime a transmission event occurs, Contagions picks from the Poisson distribution to determine the number of migrants.

Note: Text values are care sensitive.

transmission_size parameter

The `transmission_size` parameter specifies the number of pathogens to be transmitted everytime a transmission event will occur. This parameter works in concert with the `transmission_mode` parameter to determine the number of migrants.

When the `transmission_mode` parameter is set to “constant”, the `transmission_size` value is the number of pathogens to be transmitted given a transmission event. On the other hand, if the `transmission_mode` parameter is set to “poisson”, the `transmission_size` value indicates the mean and variance of the Poisson distribution from which the number of pathogen migrants are drawn.

Stop conditions

This is a special section that instructs Contagion to halt the simulation once a particular condition is reached.

condition parameter

The `condition` parameter tells Contagion under what kind condition will possibly stop the simulation. There are three conditions to choose from - allele loss, allele fixation or loss, and genotype loss.

Table 11. Available stop conditions

Keyword	Condition	Description
allele_loss	allele loss	Stops the simulation if a particular allele in a specified site becomes extinct.
allele_fixloss	allele fixation or loss	Stops the simulation if a particular allele in a specified site becomes fixed or becomes extinct.
genotype_loss	genotype loss	Stops the simulation if a particular genotype becomes extinct.

Note: Text values are care sensitive.

sequence parameter

The `sequence` parameter specifies what the character (for allele) or string (for genotype) to match against. This parameter works in concert with the `position` parameter. For example, if \`allele_loss\` is selected, the simulation will halt if the value of `sequence` is no longer found at a particular site.

For “allele_loss” and “allele_fixloss”, Contagion expects a single character. For “genotype_loss”, the program expects a string of at least one character.

position parameter

The `position` parameter specifies the position of the site to check at every generation. This parameter applies only for “allele_loss” and “allele_fixloss” conditions.

APPENDIX II
CONTAGION API

INTRODUCTION

This section contains Contagion's application programming interface (API) to facilitate users to create new functionality, edit existing processes, and customize the source code of the program to their needs. However, editing the source code is not necessary to use the program. This section is intended only for users who would like to edit or add functionality to the program. For more information about how to use the program and how to make configuration file, refer to Appendix I instead.

Contagion is written in the Go programming language. In order to edit or add new functionality, knowledge of programming in Go is necessary. For more information about the Go programming language, please refer to the "Learning Go" section in the Go website at https://golang.org/doc/.

This section is generated from comments and declarations found in the program's source code. The Contagion API is currently under development and the structure of types and structs, and parameters in functions and methods, as well as their descriptions may change. Please refer to the Contagion API webpage at https://godoc.org/github.com/kentwait/contagion for the latest API information.

PACKAGE CONTAGION

```
import "github.com/kentwait/contagion"
```

To import Contagion as a Go package, use the following import statement.

CONSTANTS

```
const (
    // GraphPathogenTypeAssertionError is the message printed when
    // a GraphPathogen cannot be asserted for an interface
    GraphPathogenTypeAssertionError = "error asserting PathogenNode interf
ace"

    InvalidFloatParameterError  = "invalid %s %f, %s"
    InvalidIntParameterError    = "invalid %s %d, %s"
    InvalidStringParameterError = "invalid %s %s, %s"
)
const (
    UnequalFloatParameterError = "expected %s %f, instead got %f"
    EqualFloatParameterError   = "%s should not be equal: %f, %f"
    FloatNotBetweenError       = "expected %s between %f and %f, instead g
ot %f"

    UnequalIntParameterError = "expected %s %d, instead got %d"
    EqualIntParameterError   = "%s should not be equal: %d, %d"
    IntNotBetweenError       = "expected %s between %d and %d, instead got
 %d"

    UnequalStringParameterError = "expected %s %s, instead got %s"
    EqualStringParameterError   = "%s should not be idenitical: %s, %s"

    UnexpectedErrorWhileError = "encountered error while %s: %s"
    ExpectedErrorWhileError   = "expected an error while %s, instead got n
one"
    UnrecognizedKeywordError  = "%s is not a valid keyword for %s"
)
const (
    IdenticalPointerError    = "memory address of %s (%p) and %s (%p) are
identical"
    NotIdenticalPointerError = "memory address of %s (%p) and %s (%p) are
not identical"
)
const (
    SusceptibleStatusCode = 1
    ExposedStatusCode     = 2
    InfectedStatusCode    = 3
    InfectiveStatusCode   = 4
    RemovedStatusCode     = 5
    RecoveredStatusCode   = 6
    DeadStatusCode        = 7
    VaccinatedStatusCode  = 8
)
```

These are status codes for different preset compartments that describe the current epidemiological status of a host in the simulation.

FUNCTIONS

func AppendToFile

```
func AppendToFile(path string, b []byte) error
```

AppendToFile creates a new file on the given path if it does not exist, or appends to the end of the existing file if the file exists.

func ConnectionDoesNotExistError

```
func ConnectionDoesNotExistError(a, b int) error
```

ConnectionDoesNotExistError indicates that a connection between hosts a and b (int) does not exist but is expected to exist.

func ConnectionExistsError

```
func ConnectionExistsError(a, b int, value float64) error
```

ConnectionExistsError indicates that a connection between the source host and the destination host exists and has the following value in float64.

func DuplicateSitePositionError

```
func DuplicateSitePositionError(pos int, lineNum int) error
```

DuplicateSitePositionError indicates that the site in the file is not unique and has been included more than once.

func DurationTooLongError

```
func DurationTooLongError(interval string, intervalDuration int, condition
 string, conditionValue int) error
```

DurationTooLongError indicates that the duration of a particular interval is longer than expected.

func DurationTooShortError

```
func DurationTooShortError(interval string, intervalDuration int, conditio
n string, conditionValue int) error
```

DurationTooShortError indicates that the duration of a particular interval is shorter than expected.

func EmptyMatrixError

```
func EmptyMatrixError() error
```

func EmptyModelError

```
func EmptyModelError() error
```

EmptyModelError indicates that a model should exist but instead is nil.

func ExchangePathogens

```
func ExchangePathogens(i, t int, h1, h2 Host, h1Count, h2Count int, c chan
<- ExchangeEvent, d chan<- TransmissionPackage, wg *sync.WaitGroup)
```

ExchangePathogens exchanges pathogens between neighboring hosts.

func Exists

```
func Exists(path string) (bool, error)
```

Exists returns whether the given file or directory exists or not, and accompanying errors.

func ExposedDurationTooLongError

```
func ExposedDurationTooLongError(intervalDuration int, conditionValue int)
 error
```

ExposedDurationTooLongError indicates that the duration in the exposed state is too long.

func ExposedDurationTooShortError

```
func ExposedDurationTooShortError(intervalDuration int, conditionValue in
t) error
```

ExposedDurationTooShortError indicates that the duration in the exposed state is too short.

func FileDoesNotExistError

```
func FileDoesNotExistError(path string) error
```

FileDoesNotExistError indicates that a file does not exist at the given path when it is expected to.

func FileExistsCheckError

```
func FileExistsCheckError(err error, path string) error
```

FileExistsCheckError indicates an error was encountered while checking if the files exists. This is not the same with an error because a file exists.

func FileExistsError

```
func FileExistsError(path string) error
```

FileExistsError indicates that a file exists at the given path.

func FileOpenError

```
func FileOpenError(err error) error
```

FileOpenError indicates that an error was encountered while opening a file.

func FileParsingError

```
func FileParsingError(err error, lineNum int) error
```

FileParsingError indicates a parsing error was encountered at a particular line in the file. Most likely the file was not properly formatted.

func FileSyncError

```
func FileSyncError(err error) error
```

FileSyncError indicates that an error was encountered while the file was being flushed from memory and being written to disk.

func FileWriteError

```
func FileWriteError(err error) error
```

FileWriteError indicates that an error was encountered while writing to the file in memory.

func InfectedDurationTooLongError

```
func InfectedDurationTooLongError(intervalDuration int, conditionValue in
t) error
```

InfectedDurationTooLongError indicates that the duration in the infected state is too long.

func InfectedDurationTooShortError

```
func InfectedDurationTooShortError(intervalDuration int, conditionValue in
t) error
```

InfectedDurationTooShortError indicates that the duration in the infected state is too short.

func InfectiveDurationTooLongError

```
func InfectiveDurationTooLongError(intervalDuration int, conditionValue in
t) error
```

InfectiveDurationTooLongError indicates that the duration in the infective state is too long.

func InfectiveDurationTooShortError

```
func InfectiveDurationTooShortError(intervalDuration int, conditionValue i
nt) error
```

InfectiveDurationTooShortError indicates that the duration in the infective state is too short.

func IntKeyExists

```
func IntKeyExists(key int) error
```

IntKeyExists indicates that the given integer key already exists.

func IntKeyNotFoundError

```
func IntKeyNotFoundError(key int) error
```

IntKeyNotFoundError indicates that the given integer key does not exist.

func IntrinsicRateReplication

```
func IntrinsicRateReplication(pathogens []GenotypeNode, replFitness []floa
t64, immuneSystem interface{}) <-chan GenotypeNode
```

IntrinsicRateReplication replicates pathogens by considering their fitness value as the growth rate.

func InvalidCharError

```
func InvalidCharError(pos int, enc uint8) error
```

func InvalidConnectionWeightError

```
func InvalidConnectionWeightError(wt float64, lineNum int) error
```

InvalidConnectionWeightError indicates that the given connection weight is less than 0.

func InvalidRowError

```
func InvalidRowError() error
```

func InvalidStateCharError

```
func InvalidStateCharError(char string, pos int) error
```

InvalidStateCharError indicates that a character encountered is not in the set of expected characters.

func LoadFitnessMatrix

```
func LoadFitnessMatrix(path string, valueType string) (map[int]map[uint8]f
loat64, error)
```

LoadFitnessMatrix parses and loads the fitness matrix encoded in the text file at the given path.

func LoadSequences

```
func LoadSequences(path string) (map[int][][]uint8, error)
```

LoadSequences parses a specially-formatted FASTA file to get the sequences, encode sequences into integers, and distribute to assigned hosts. Returns a map where the key is the host ID and the values are the pathogen sequences for the particular host.

func ModelExistsError

```
func ModelExistsError(modelName string, modelID int) error
```

ModelExistsError indicates that an existing model already exists a new model cannot be assigned to replace it.

func MotifExistsError

```
func MotifExistsError(motifID string) error
```

MotifExistsError indicates that a motif with the same sequence and positions already exists in the model.

func MultinomialReplication

```
func MultinomialReplication(pathogens []GenotypeNode, normedFitnesses []fl
oat64, newPopSize int) <-chan GenotypeNode
```

MultinomialReplication replicates and selects sequences based on normalized fitness values used as probabilities.

func MutateSequence

```
func MutateSequence(sequences <-chan GenotypeNode, tree GenotypeTree, mode
l IntrahostModel) (<-chan GenotypeNode, <-chan GenotypeNode)
```

MutateSequence adds substitution mutations to sequenceNode.

func MutateSite

```
func MutateSite(transitionProbs ...float64) uint8
```

MutateSite returns the new state of a site based on the given a set of transition probabilities.

func NewFile

```
func NewFile(path string, b []byte) error
```

NewFile creates a new file on the given path if it does not exist. Returns an error if the file exists.

func OpenSQLiteDB

```
func OpenSQLiteDB(path, connectionString string) (*sql.DB, error)
```

OpenSQLiteDB establishes a database connection using the given connection string.

func OpenSQLiteDBOptimized

```
func OpenSQLiteDBOptimized(path string) (*sql.DB, error)
```

OpenSQLiteDBOptimized establishes a database connection using WAL and exclusive locking.

func OverlappingMotifError

```
func OverlappingMotifError(pos int) error
```

OverlappingMotifError indicates that a particular site cannot be used again because it is already being considered by another motif in the model.

func RecombineAnySequence

```
func RecombineAnySequence(numSeqs, numRecSites int, sequences <-chan Genot
ypeNode, tree GenotypeTree, model IntrahostModel) (<-chan GenotypeNode, <-
chan GenotypeNode)
```

RecombineAnySequence recombines any two sequences at a random position similar to the behavior of template switching.

func RecombineSequencePairs

```
func RecombineSequencePairs(numSeqs, numRecSites int, sequences <-chan Gen
otypeNode, tree GenotypeTree, model IntrahostModel) (<-chan GenotypeNode,
<-chan GenotypeNode)
```

RecombineSequencePairs recombines two sequences at random positions similar to the behavior of diploid chromosomes.

func RemovedDurationTooLongError

```
func RemovedDurationTooLongError(intervalDuration int, conditionValue int)
 error
```

RemovedDurationTooLongError indicates that the duration in the removed state is too long.

func RemovedDurationTooShortError

```
func RemovedDurationTooShortError(intervalDuration int, conditionValue in
t) error
```

RemovedDurationTooShortError indicates that the duration in the removed state is too short.

func SQLBeginTransactionError

```
func SQLBeginTransactionError(err error) error
```

SQLBeginTransactionError indicates that an error was encountered while a transaction was being initialized.

func SQLExecError

```
func SQLExecError(err error, stmt string) error
```

SQLExecError indicates that an error was encountered while executing an SQL statement. Returns the error raised by the database connection and the SQL statement that produced the error.

func SQLExecStatementError

```
func SQLExecStatementError(err error) error
```

SQLExecStatementError indicates that an error was encountered while a template statement was being substituted with actual values.

func SQLOpenError

```
func SQLOpenError(err error) error
```

SQLOpenError indicates that an error was encountered while open a database connection. Includes the error returned by sql.Open.

func SQLPrepareStatementError

```
func SQLPrepareStatementError(err error, stmt string) error
```

SQLPrepareStatementError indicates that an error was encountered while a template SQL statement was being initialized.

func SelfLoopError

```
func SelfLoopError(hostID int) error
```

SelfLoopError indicates that the start and end host are the same based on host ID, which results in a self-loop.

func SetFitnessModelExistsError

```
func SetFitnessModelExistsError(modelName string, modelID int) error
```

SetFitnessModelExistsError indicates that a fitness model has already been assigned to a host.

func SetIntrahostModelExistsError

```
func SetIntrahostModelExistsError(modelName string, modelID int) error
```

SetIntrahostModelExistsError indicates that an intrahost model has already been assigned to a host.

func SetTransmissionModelExistsError

```
func SetTransmissionModelExistsError(modelName string, modelID int) error
```

SetTransmissionModelExistsError indicates that a transmission model has already been assigned to a host.

func TransmitPathogens

```
func TransmitPathogens(i, t int, src, dst Host, numMigrants int, transmiss
ionProb float64, count int, c chan<- TransmissionEvent, d chan<- Transmiss
ionPackage, wg *sync.WaitGroup)
```

TransmitPathogens transmits the pathogen to its neighboring host/s. If transmission occurs, sends transmitted node over the channel to be added to the recepient. Also sends node information in order to record the event.

func UnequalNumStatesError

```
func UnequalNumStatesError(numStates, prevNumStates int, site int, lineNum
 int) error
```

UnequalNumStatesError indicates that the number of states specified in a site does not match the number of states in another site.

func ZeroItemsError

```
func ZeroItemsError() error
```

ZeroItemsError indicates that the length of a list or set is empty but at least one item is required.

type BevertonHoltThresholdPopModel

```
type BevertonHoltThresholdPopModel struct {
    // contains filtered or unexported fields
}
```

BevertonHoltThresholdPopModel uses the Beverton-Holt population model modified to have a constant threshold population size.

func (*BevertonHoltThresholdPopModel) GrowthRate

```
func (m *BevertonHoltThresholdPopModel) GrowthRate() float64
```

func (*BevertonHoltThresholdPopModel) MaxPathogenPopSize

```
func (m *BevertonHoltThresholdPopModel) MaxPathogenPopSize() int
```

func (*BevertonHoltThresholdPopModel) ModelID

```
func (meta *BevertonHoltThresholdPopModel) ModelID() int
```

func (*BevertonHoltThresholdPopModel) ModelName

```
func (meta *BevertonHoltThresholdPopModel) ModelName() string
```

*func (*BevertonHoltThresholdPopModel) MutationRate*

```
func (params *BevertonHoltThresholdPopModel) MutationRate() float64
```

*func (*BevertonHoltThresholdPopModel) NextPathogenPopSize*

```
func (m *BevertonHoltThresholdPopModel) NextPathogenPopSize(n int) int
```

*func (*BevertonHoltThresholdPopModel) ProbabilisticDuration*

```
func (params *BevertonHoltThresholdPopModel) ProbabilisticDuration() bool
```

*func (*BevertonHoltThresholdPopModel) RecombinationRate*

```
func (params *BevertonHoltThresholdPopModel) RecombinationRate() float64
```

*func (*BevertonHoltThresholdPopModel) ReplicationMethod*

```
func (m *BevertonHoltThresholdPopModel) ReplicationMethod() string
```

*func (*BevertonHoltThresholdPopModel) SetModelID*

```
func (meta *BevertonHoltThresholdPopModel) SetModelID(id int)
```

*func (*BevertonHoltThresholdPopModel) SetModelName*

```
func (meta *BevertonHoltThresholdPopModel) SetModelName(name string)
```

func (*BevertonHoltThresholdPopModel) StatusDuration

```
func (params *BevertonHoltThresholdPopModel) StatusDuration(status int) in
t
```

func (*BevertonHoltThresholdPopModel) TransitionMatrix

```
func (params *BevertonHoltThresholdPopModel) TransitionMatrix() [][]float6
4
```

func (*BevertonHoltThresholdPopModel) TransitionProbs

```
func (params *BevertonHoltThresholdPopModel) TransitionProbs(char int) []f
loat64
```

type CSVLogger

```
type CSVLogger struct {
    // contains filtered or unexported fields
}
```

CSVLogger is a DataLogger that writes simulation data as comma-delimited files.

func NewCSVLogger

```
func NewCSVLogger(basepath string, i int) *CSVLogger
```

NewCSVLogger creates a new logger that writes data into CSV files.

func (*CSVLogger) Init

```
func (l *CSVLogger) Init() error
```

Init creates CSV files and writes header information for each file.

func (*CSVLogger) SetBasePath

```
func (l *CSVLogger) SetBasePath(basepath string, i int)
```

SetBasePath sets the base path of the logger.

func (*CSVLogger) WriteGenotypeFreq

```
func (l *CSVLogger) WriteGenotypeFreq(c <-chan GenotypeFreqPackage)
```

WriteGenotypeFreq records the count of unique genotype nodes present within the host in a given time in the simulation.

func (*CSVLogger) WriteGenotypeNodes

```
func (l *CSVLogger) WriteGenotypeNodes(c <-chan GenotypeNode)
```

WriteGenotypeNodes records new genotype node's ID and associated genotype ID to file

func (*CSVLogger) WriteGenotypes

```
func (l *CSVLogger) WriteGenotypes(c <-chan Genotype)
```

WriteGenotypes records a new genotype's ID and sequence to file.

*func (*CSVLogger) WriteMutations*

```
func (l *CSVLogger) WriteMutations(c <-chan MutationPackage)
```

WriteMutations records every time a new genotype node is created. It records the time and in what host this new mutation arose.

*func (*CSVLogger) WriteStatus*

```
func (l *CSVLogger) WriteStatus(c <-chan StatusPackage)
```

WriteStatus records the status of each host every generation.

*func (*CSVLogger) WriteTransmission*

```
func (l *CSVLogger) WriteTransmission(c <-chan TransmissionPackage)
```

WriteTransmission records the ID's of genotype node that are transmitted between hosts.

type Config

```
type Config interface {
    Validate() error
    NewSimulation() (Epidemic, error)
    NumInstances() int
    NumGenerations() int
    LogFreq() int
    LogPath() string
    LogTransmission() bool

}
```

Config represents any top level TOML configuration that can create a new simulation.

type ConstantPopModel

```
type ConstantPopModel struct {
    // contains filtered or unexported fields

}
```

ConstantPopModel models a constant pathogen population size within the host.

*func (*ConstantPopModel) MaxPathogenPopSize*

```
func (m *ConstantPopModel) MaxPathogenPopSize() int
```

*func (*ConstantPopModel) ModelID*

```
func (meta *ConstantPopModel) ModelID() int
```

*func (*ConstantPopModel) ModelName*

```
func (meta *ConstantPopModel) ModelName() string
```

*func (*ConstantPopModel) MutationRate*

```
func (params *ConstantPopModel) MutationRate() float64
```

*func (*ConstantPopModel) NextPathogenPopSize*

```
func (m *ConstantPopModel) NextPathogenPopSize(n int) int
```

*func (*ConstantPopModel) ProbabilisticDuration*

```
func (params *ConstantPopModel) ProbabilisticDuration() bool
```

func (*ConstantPopModel) RecombinationRate

```
func (params *ConstantPopModel) RecombinationRate() float64
```

func (*ConstantPopModel) ReplicationMethod

```
func (m *ConstantPopModel) ReplicationMethod() string
```

func (*ConstantPopModel) SetModelID

```
func (meta *ConstantPopModel) SetModelID(id int)
```

func (*ConstantPopModel) SetModelName

```
func (meta *ConstantPopModel) SetModelName(name string)
```

func (*ConstantPopModel) StatusDuration

```
func (params *ConstantPopModel) StatusDuration(status int) int
```

func (*ConstantPopModel) TransitionMatrix

```
func (params *ConstantPopModel) TransitionMatrix() [][]float64
```

func (*ConstantPopModel) TransitionProbs

```
func (params *ConstantPopModel) TransitionProbs(char int) []float64
```

type DataLogger

```
type DataLogger interface {
    // SetBasePath sets the base path of the logger.
    SetBasePath(path string, i int)
    // Init initializes the logger. For example, if the logger writes a

    // CSV file, Init can create a file and write header information 
first.
    // Or if the logger writes to a database, Init can be used to
    // create a new table.
    Init() error
    // WriteGenotypes records a new genotype's ID and sequence to file.
    WriteGenotypes(c <-chan Genotype)
    // WriteGenotypeNodes records new genotype node's ID and
    // associated genotype ID to file
    WriteGenotypeNodes(c <-chan GenotypeNode)
    // WriteGenotypeFreq records the count of unique genotype nodes
    // present within the host in a given time in the simulation.
    WriteGenotypeFreq(c <-chan GenotypeFreqPackage)
    // WriteMutations records every time a new genotype node is created.
    // It records the time and in what host this new mutation arose.
    WriteMutations(c <-chan MutationPackage)
    // WriteStatus records the status of each host every generation.
    WriteStatus(c <-chan StatusPackage)
    // WriteTransmission records the ID's of genotype node that
    // are transmitted between hosts.
    WriteTransmission(c <-chan TransmissionPackage)

}
```

DataLogger is the general definition of a logger that records simulation data to file whether it writes a text file or writes to a database.

type EndTransSimulation

```
type EndTransSimulation struct {
    EpidemicSimulation

}
```

EndTransSimulation creates and runs a modified version of the SIR epidemiological simulation. In the endtrans model, transmissions are allowed to occur only at the last generation before pathogens are cleared from the host. To make transmission completely deterministic, set the transmission probability to 1.0, use the constant mode and set it to a constant size. This means that all paths connected to an infectable host gets infected at the end of the infection cycle of the current host. Endtrans assumes that the InfectedDuration is not zero.

func NewEndTransSimulation

```
func NewEndTransSimulation(config Config, logger DataLogger) (*EndTransSim
ulation, error)
```

NewEndTransSimulation creates a new SI simulation.

func (*EndTransSimulation) Run

```
func (sim *EndTransSimulation) Run(i int)
```

Run instantiates, runs, and records the a new simulation.

func (*EndTransSimulation) Transmit

```
func (sim *EndTransSimulation) Transmit(t int)
```

Transmit facilitates the sampling and migration process of pathogens between hosts.

*func (*EndTransSimulation) Update*

```
func (sim *EndTransSimulation) Update(t int)
```

type Epidemic

```
type Epidemic interface {
    // Host returns the selected host in the simulation.
    Host(id int) Host
    // HostStatus retrieves the current status of the selected host.
    HostStatus(id int) int
    // SetHostStatus sets the current status of the selected host
    // to a given status code.
    SetHostStatus(id, status int)
    // HostTimer returns the current number of generations remaining
    // before the host changes status.
    HostTimer(id int) int
    // SetHostTimer sets the number of generations for the host to
    // remain in its current status.
    SetHostTimer(id, interval int)
    // InfectableStatuses returns the list of statuses that infected
    // hosts can transmit to.
    InfectableStatuses() []int


    // HostMap returns the hosts in the simulation in the form of a map.
    // The key is the host's ID and the value is the pointer to the host.
    HostMap() map[int]Host
    // HostConnection returns the weight of a connection between two hosts
    // if it exists, returns 0 otherwise.
    HostConnection(a, b int) float64
    // HostNeighbors retrieves the directly connected hosts to the current
    // host based on the supplied adjacency matrix.
    HostNeighbors(id int) []Host
    // NewInstance creates a new instance from the stored configuration
    NewInstance() (Epidemic, error)


    // GenotypeNodeMap returns the set of all GenotypeNodes seen since the
    // start of the simulation.
    GenotypeNodeMap() map[ksuid.KSUID]GenotypeNode


    // GenotypeSet returns the set of all Genotypes seen since the
    // start of the simulation.
    GenotypeSet() GenotypeSet


    // StopSimulation check whether the simulation has satisfied at least
one
    // of the conditions that will halt the simulation in the current inte
ration.
    StopSimulation() bool


    // SusceptibleProcess performs intrahost processes while the host is i
n
    // the susceptible status.
    SusceptibleProcess(i, t int, host Host, wg *sync.WaitGroup)
    // ExposedProcess performs intrahost processes while the host is in
    // the exposed status.
    ExposedProcess(i, t int, host Host, c chan<- MutationPackage, wg *syn
c.WaitGroup)
    // InfectedProcess performs intrahost processes while the host is in
    // the infected status.
    InfectedProcess(i, t int, host Host, c chan<- MutationPackage, wg *syn
c.WaitGroup)
    // InfectiveProcess performs intrahost processes while the host is in
    // the infective status.
    InfectiveProcess(i, t int, host Host, c chan<- MutationPackage, wg *sy
nc.WaitGroup)
    // RemovedProcess performs intrahost processes while the host is in
    // the removed status.
    RemovedProcess(i, t int, host Host, wg *sync.WaitGroup)
    // RecoveredProcess performs intrahost processes while the host is in
    // the recovered status.
    RecoveredProcess(i, t int, host Host, wg *sync.WaitGroup)
    // DeadProcess performs intrahost processes while the host is in
    // the dead status.
    DeadProcess(i, t int, host Host, wg *sync.WaitGroup)
    // DeadProcess performs intrahost processes while the host is in
    // the dead status.
    VaccinatedProcess(i, t int, host Host, wg *sync.WaitGroup)

}
```

Epidemic encapsulates the set of hosts, its connections, the pathogen tree lineage and the host types used to create a simulated epidemic.

type EpidemicSimulation

```
type EpidemicSimulation interface {
    Epidemic
    DataLogger


    // Run runs the whole simulation

    Initialize(params ...interface{})
    Run(i int)
    Update(t int)
    Process(t int)
    Transmit(t int)

    Finalize()

    // Metadata
    SetInstanceID(i int)
    InstanceID() int
    SetTime(t int)
    Time() int
    SetGenerations(n int)
    NumGenerations() int
    LogTransmission() bool
    LogFrequency() int
    SetStopped(b bool)
    Stopped() bool

}
```

EpidemicSimulation is a simulation environment that simulates the spread of disease between hosts in a connected host network.

type EvoEpiConfig

```
type EvoEpiConfig struct {
    SimParams          *epidemicSimConfig      `toml:"simulation"`
    LogParams          *logConfig              `toml:"logging"`
    IntrahostModels    []*intrahostModelConfig `toml:"intrahost_model"`
    FitnessModels      []*fitnessModelConfig   `toml:"fitness_model"`
    TransmissionModels []*transModelConfig     `toml:"transmission_model"`
    StopConditions     []*stopConditionConfig  `toml:"stop_condition"`
    // contains filtered or unexported fields

}
```

EvoEpiConfig contains parameters to create a simulated infection in a connected network of hosts.

func LoadEvoEpiConfig

```
func LoadEvoEpiConfig(path string) (*EvoEpiConfig, error)
```

LoadEvoEpiConfig creates an EvoEpiConfig struct from a TOML file.

func (*EvoEpiConfig) LogFreq

```
func (c *EvoEpiConfig) LogFreq() int
```

LogFreq returns the number of pathogen generations in the simulation until data is recorded.

func (*EvoEpiConfig) LogPath

```
func (c *EvoEpiConfig) LogPath() string
```

LogPath returns the path where to write results. This can a path to a folder, or directory_path/prefix format

func (*EvoEpiConfig) LogTransmission

```
func (c *EvoEpiConfig) LogTransmission() bool
```

LogTransmission indicates whether transmissions are logged or discarded.

func (*EvoEpiConfig) NewSimulation

```
func (c *EvoEpiConfig) NewSimulation() (Epidemic, error)
```

NewSimulation creates a new SingleHostSimulation simulation.

func (*EvoEpiConfig) NumGenerations

```
func (c *EvoEpiConfig) NumGenerations() int
```

NumGenerations returns the number of pathogen generation in a single simulation run.

*func (*EvoEpiConfig) NumInstances*

```
func (c *EvoEpiConfig) NumInstances() int
```

NumInstances returns the number of independent realizations to run.

*func (*EvoEpiConfig) Validate*

```
func (c *EvoEpiConfig) Validate() error
```

Validate checks the validity of the configuration.

type ExchangeEvent

```
type ExchangeEvent struct {
    // contains filtered or unexported fields

}
```

ExchangeEvent is a struct for sending and receiving exchange event information.

type ExchangeSimulation

```
type ExchangeSimulation struct {
    SISimulation
    // contains filtered or unexported fields

}
```

ExchangeSimulation creates and runs a modified version of the SIR epidemiological simulation. In ExchangeSimulation, all hosts are initially infected.

func NewExchangeSimulation

```
func NewExchangeSimulation(config Config, logger DataLogger) (*ExchangeSim
ulation, error)
```

NewExchangeSimulation creates a new migration simulation.

func (*ExchangeSimulation) Process

```
func (sim *ExchangeSimulation) Process(t int)
```

Process runs the internal evolution simulation in each host. During intrahost evolution, if new mutations appear, the new sequence and ancestry is recorded to file.

func (*ExchangeSimulation) Run

```
func (sim *ExchangeSimulation) Run(i int)
```

Run instantiates, runs, and records the a new simulation.

func (*ExchangeSimulation) Transmit

```
func (sim *ExchangeSimulation) Transmit(t int)
```

Transmit facilitates the sampling and migration process of pathogens between hosts.

func (*ExchangeSimulation) Update

```
func (sim *ExchangeSimulation) Update(t int)
```

Update looks at the timer or internal state to decide if the status of the host remains the same of will change. After the status updates, each host's status is recorded to file.

type FitnessDependentPopModel

```
type FitnessDependentPopModel struct {
    // contains filtered or unexported fields

}
```

FitnessDependentPopModel does not use a population model to determine the population of pathogens. Instead population size is dependent on fitness which is implemented outside of this model. The NextPathogenPopSize method for this model always returns -1 regardless of the input value.

func (*FitnessDependentPopModel) MaxPathogenPopSize

```
func (m *FitnessDependentPopModel) MaxPathogenPopSize() int
```

func (*FitnessDependentPopModel) ModelID

```
func (meta *FitnessDependentPopModel) ModelID() int
```

func (*FitnessDependentPopModel) ModelName

```
func (meta *FitnessDependentPopModel) ModelName() string
```

func (*FitnessDependentPopModel) MutationRate

```
func (params *FitnessDependentPopModel) MutationRate() float64
```

func (*FitnessDependentPopModel) NextPathogenPopSize

```
func (m *FitnessDependentPopModel) NextPathogenPopSize(n int) int
```

func (*FitnessDependentPopModel) ProbabilisticDuration

```
func (params *FitnessDependentPopModel) ProbabilisticDuration() bool
```

func (*FitnessDependentPopModel) RecombinationRate

```
func (params *FitnessDependentPopModel) RecombinationRate() float64
```

func (*FitnessDependentPopModel) ReplicationMethod

```
func (m *FitnessDependentPopModel) ReplicationMethod() string
```

func (*FitnessDependentPopModel) SetModelID

```
func (meta *FitnessDependentPopModel) SetModelID(id int)
```

func (*FitnessDependentPopModel) SetModelName

```
func (meta *FitnessDependentPopModel) SetModelName(name string)
```

func (*FitnessDependentPopModel) StatusDuration

```
func (params *FitnessDependentPopModel) StatusDuration(status int) int
```

func (*FitnessDependentPopModel) TransitionMatrix

```
func (params *FitnessDependentPopModel) TransitionMatrix() [][]float64
```

*func (*FitnessDependentPopModel) TransitionProbs*

```
func (params *FitnessDependentPopModel) TransitionProbs(char int) []float6
4
```

type FitnessMatrix

```
type FitnessMatrix interface {
    // ID returns the ID for this fitness model.
    ModelID() int
    // Name returns the name for this fitness model.
    ModelName() string
    SetModelID(id int)
    SetModelName(name string)
    // ComputeFitness returns the corresponding fitness value given
    // a set of sequences as integers.
    ComputeFitness(chars ...uint8) (fitness float64, err error)
    // SiteFitness returns the fitness value associated for a particular
    // character at the given site.
    SiteCharFitness(position int, state uint8) (fitness float64, err erro
r)
    // Log tells whether the fitness values are decimal or log.
    // Usually fitness is in log.
    Log() bool

}
```

FitnessMatrix is a type of FitnessModel where the fitness of each individual character at every site is specified.

func NeutralAdditiveFM

```
func NeutralAdditiveFM(id int, name string, sites, alleles, growthRate in
t) (FitnessMatrix, error)
```

NeutralAdditiveFM returns a additive fitness matrix where the sum of all sites using any allele combination is equal to the growth rate.

func NeutralMultiplicativeFM

```
func NeutralMultiplicativeFM(id int, name string, sites, alleles int) (Fit
nessMatrix, error)
```

NeutralMultiplicativeFM returns a multiplicative fitness matrix where all the values are 0 (ln 1) such that all changes have no effect and are therefore neutral.

func NewAdditiveFM

```
func NewAdditiveFM(id int, name string, matrix map[int]map[uint8]float64)
(FitnessMatrix, error)
```

NewAdditiveFM create a new additive fitness matrix using a map of maps. Assumes that the values are in decimal form.

func NewMultiplicativeFM

```
func NewMultiplicativeFM(id int, name string, matrix map[int]map[uint8]flo
at64) (FitnessMatrix, error)
```

NewMultiplicativeFM create a new multiplicative fitness matrix using a map of maps. Assumes that the values are in log form.

type FitnessModel

```
type FitnessModel interface {
    // ID returns the ID for this fitness model.
    ModelID() int
    // Name returns the name for this fitness model.
    ModelName() string
    SetModelID(id int)
    SetModelName(name string)
    // ComputeFitness returns the corresponding fitness value given
    // a set of sequences as integers.
    ComputeFitness(chars ...uint8) (fitness float64, err error)

}
```

FitnessModel represents a general method to determine the fitness value associated to a particular genotype.

type Genotype

```
type Genotype interface {
    GenotypeUID() ksuid.KSUID
    // Sequence returns the sequence of the current node.
    Sequence() []uint8
    // SetSequence changes the sequence of genotype.
    SetSequence(sequence []uint8)
    // StringSequence returns the string representation of the
    // integer-coded sequence of the current node.
    StringSequence() string
    // Fitness returns the fitness value of this node based on its current

    // sequence and the given fitness model. If the fitness of the node
has
    // been computed before using the same fitness model, then the value
is
    // returned from memory and is not recomputed.
    Fitness(f FitnessModel) float64
    // NumSites returns the number of sites being modeled in this pathogen
 node.
    NumSites() int
    // StateCounts returns the number of sites by state.
    StateCounts() map[uint8]int
    // StatePositions returns the indexes of sites in a particular state.
    StatePositions(state uint8) []int

}
```

Genotype represents a unique pathogen sequence.

func NewGenotype

```
func NewGenotype(s []uint8) Genotype
```

NewGenotype creates a new genotype from sequence.

type GenotypeFreqPackage

```
type GenotypeFreqPackage struct {
    // contains filtered or unexported fields

}
```

GenotypeFreqPackage encapsulates the data to be written everytime the frequency of genotypes have to be recorded.

type GenotypeNode

```
type GenotypeNode interface {
    // UID returns the unique ID of the node. Uses KSUID to generate
    // random unique IDs with effectively no collision.
    UID() ksuid.KSUID
    GenotypeUID() ksuid.KSUID
    // Parents returns the parent of the node.
    Parents() []GenotypeNode
    // Children returns the children of the node.
    Children() []GenotypeNode
    // AddChild appends a child to the list of children.
    AddChild(child GenotypeNode)
    // Sequence returns the sequence of the current node.
    Sequence() []uint8
    // SetSequence changes the sequence of genotype.
    SetSequence(sequence []uint8)
    // StringSequence returns the string representation of the
    // integer-coded sequence of the current node.
    StringSequence() string
    // CurrentGenotype returns the current genotype of the current node.
    CurrentGenotype() Genotype
    // History returns the list of sequences that resulted into the extant
    // sequence.
    History(h [][]uint8) [][]uint8
    // Fitness returns the fitness value of this node based on its current

    // sequence and the given fitness model. If the fitness of the node
has
    // been computed before using the same fitness model, then the value
is
    // returned from memory and is not recomputed.
    Fitness(f FitnessModel) float64
    // NumSites returns the number of sites being modeled in this pathogen
 node.
    NumSites() int
    // StateCounts returns the number of sites by state.
    StateCounts() map[uint8]int
    // StatePositions returns the indexes of sites in a particular state.
    StatePositions(state uint8) []int

}
```

GenotypeNode represents a genotype together with its relationship to its parents and children.

type GenotypeSet

```
type GenotypeSet interface {
    // Add adds the genotype to the set if the sequence does not exist ye
t.
    Add(g Genotype)
    // AddSequence creates a new genotype from the sequence if it is not p
resent
    // in the set. Otherwise, returns the existing genotype in the set.
    AddSequence(s []uint8) Genotype
    // Remove removes genotype of a particular sequence from the set.
    Remove(s []uint8)
    // Size returns the size of the set.
    Size() int
    Map() map[string]Genotype

}
```

GenotypeSet is a collection of genotypes.

func EmptyGenotypeSet

```
func EmptyGenotypeSet() GenotypeSet
```

EmptyGenotypeSet creates a new empty set.

type GenotypeTree

```
type GenotypeTree interface {
    // Set returns the GenotypeSet associated with this tree.
    Set() GenotypeSet
    // NewNode creates a new genotype node from a given sequence.
    // Automatically adds sequence to the genotypeSet if it is not yet pre
sent.
```

```
    NewNode(sequence []uint8, subs int, parents ...GenotypeNode) GenotypeN
ode
    NewRecombinantNode(sequence []uint8, recombs int, parents ...GenotypeN
ode) GenotypeNode
    // Nodes returns the map of genotype node ID found in the tree to its
    // corresponding genotype.
    NodeMap() map[ksuid.KSUID]GenotypeNode

}
```

GenotypeTree represents the genotypes as a series of differences from its ancestor.

func EmptyGenotypeTree

```
func EmptyGenotypeTree() GenotypeTree
```

EmptyGenotypeTree creates a new empty genotype tree.

type Host

```
type Host interface {
    // ID returns the unique ID of the host.
    ID() int
    // TypeID returns the ID representing the host's type in a multi-host

    // simulation. Generally, the host type ID is used to identify hosts
    // belonging to the same group that share the same properties.
    TypeID() int
    // PickPathogens returns a random list of pathogens from the

    // current host.
    // Returns nil if no pathogen exists.
    PickPathogens(n int) []GenotypeNode
    // Pathogens returns a list of all pathogens present in the host.
    // This elements of the list are pointers to GenotypeNodes.
    Pathogens() []GenotypeNode
    // PathogenPopSize returns the number of pathogens inside the host.
    PathogenPopSize() int
    // AddPathogens appends a pathogen to the pathogen space of the host.
    // Returns the new pathogen population size.
    AddPathogens(p ...GenotypeNode) int
    // RemoveAllPathogens removes all the pathogens from the host.
    // Internally, this removes all the pointers that refer to GenotypeNod
es.

    RemoveAllPathogens()
    // SetIntrahostModel associates the current host to a given intrahost
model.

    // The intrahost model governs intrahost processes by specifying the
    // mutation, replication, recombination, and infection modes and
parameters
    // to be used.
    SetIntrahostModel(intrahostModel IntrahostModel) error
    // SetFitnessModel associates the current host to a given fitness mode
l.
    SetFitnessModel(fitnessModel FitnessModel) error
    // SetTransmissionModel associates the current host to a given

    // transmission model. This model sets the transmission probability
    // (if not set in the adjacency matrix) and the transmission size.
    SetTransmissionModel(transmissionModel TransmissionModel) error
    // GetIntrahostModel retrieves the associated intrahost model
    // for the current host.
    GetIntrahostModel() IntrahostModel
    // GetFitnessModel retrieves the associated fitness model
    // for the current host.
    GetFitnessModel() FitnessModel
    // GetTransmissionModel retrieves the associated transmission model
    // for the current host.
    GetTransmissionModel() TransmissionModel

}
```

Host encapsulates pathogens together and ties its evolution to a particular set of parameters given by its assigned host type.

func EmptySequenceHost

```
func EmptySequenceHost(ids ...int) Host
```

EmptySequenceHost creates a new host without an intrahost model and no pathogens.

type HostNetwork

```
type HostNetwork interface {
    // ConnectedPopSize returns the total number of hosts in the network.
    ConnectedPopSize() int
    // GetNeighbors retrieves the unordered list of neighbors from
    // the adjacency matrix.
    GetNeighbors(ID int) (neighbors []int)
    // ConnectionExists checks if a connection a-b exists in the adjacency
 matrix.
    ConnectionExists(a, b int) bool
    // AddConnection adds a one way connection a-b to the adjacency matrix
 if

    // the connection a-b does not exists. Returns an error if given
connection
    // already exists.
    AddConnection(a, b int) error
    // AddWeightedConnection adds a one way connection a-b to the adjacenc
y
    // matrix with a given weight w.
    AddWeightedConnection(a, b int, w float64) error
    // UpdateConnectionWeight changes the weight value of an existing

    // connection. If the given connection does not exist, nothing is
    // updated.
    UpdateConnectionWeight(a, b int, w float64) error
    // UpsertConnectionWeight changes the weight value of an existing

    // connection or creates a new connection with the given weight if the
    // connection does not exist.
    UpsertConnectionWeight(a, b int, w float64)


    // AddWeightedBiConnection adds a two way reciprocal connection to the
    // adjacency matrix with a given weight.
    AddWeightedBiConnection(a, b int, w float64) error


    // DeleteConnection removes a one way connection a-b.
    DeleteConnection(a, b int) error


    // Copy returns a new copy of the adjacency matrix.

    // Changes made to the original copy will not affect the new copy
    // and changes made to the copy will likewise not affect the original.

    Copy() adjacencyMatrix


    // Dump serialized the adjacency matrix into a string stored as a byte
slice.
    Dump() []byte


    // Connection returns the weight of a connection is it exists,
    // returns 0 otherwise.
    Connection(a, b int) float64

}
```

HostNetwork interface describes a host population connected together as a network.

func EmptyAdjacencyMatrix

```
func EmptyAdjacencyMatrix() HostNetwork
```

EmptyAdjacencyMatrix creates a new 2D mapping with no contents.

func LoadAdjacencyMatrix

```
func LoadAdjacencyMatrix(path string) (HostNetwork, error)
```

LoadAdjacencyMatrix creates a new 2D mapping based on a text file.

type Infection

```
type Infection interface {
    // Host returns the selected host in the simulation.
    Host() Host


    // HostStatus retrieves the current status of the selected host.
    HostStatus() int
    // SetHostStatus sets the current status of the selected host
    // to a given status code.
    SetHostStatus(status int)
    // HostStatusDuration returns the number of generations a host
    // remains in a given status.
    HostStatusDuration(status int) int
    // HostTime returns the current number of generations remaining
    // before the host changes status.
    HostTimer() int
    // SetHostTimer sets the number of generations for the host to
    // remain in its current status.
    SetHostTimer(interval int)


    // SusceptibleProcess performs intrahost processes while the host is i
n
    // the susceptible status.
    SusceptibleProcess(i, t int, host Host, wg *sync.WaitGroup)
    // ExposedProcess performs intrahost processes while the host is in
    // the exposed status.
    ExposedProcess(i, t int, host Host, c chan<- MutationPackage, wg *syn
c.WaitGroup)
    // InfectedProcess performs intrahost processes while the host is in
    // the infected status.
    InfectedProcess(i, t int, host Host, c chan<- MutationPackage, wg *syn
c.WaitGroup)
    // InfectiveProcess performs intrahost processes while the host is in
    // the infective status.
    InfectiveProcess(i, t int, host Host, c chan<- MutationPackage, wg *sy
nc.WaitGroup)
    // RemovedProcess performs intrahost processes while the host is in
    // the removed status.
    RemovedProcess(i, t int, host Host, wg *sync.WaitGroup)
    // RecoveredProcess performs intrahost processes while the host is in
    // the recovered status.
    RecoveredProcess(i, t int, host Host, wg *sync.WaitGroup)
    // DeadProcess performs intrahost processes while the host is in
    // the dead status.
    DeadProcess(i, t int, host Host, wg *sync.WaitGroup)
    // DeadProcess performs intrahost processes while the host is in
    // the dead status.
    VaccinatedProcess(i, t int, host Host, wg *sync.WaitGroup)

}
```

Infection encapsulates a single host and the pathogen tree lineage to trace the evolution of pathogens within one host. This is useful to strudy intrahost evolutionary dynamics especially in chronic diseases.

type InfectionSimulation

```
type InfectionSimulation interface {
    Infection
    // Run runs the whole simulation

    Init(params ...interface{})
    Run(i int)
    Update(t int)
    Process(t int)
    Transmit(t int)

}
```

InfectionSimulation is a simulation environment that simulates the infection within a single hosts or in an evironment where a network configuration is not necessary.

type IntrahostModel

```
type IntrahostModel interface {
    ModelID() int
    ModelName() string
    SetModelID(id int)
    SetModelName(name string)


    // MutationRate returns the mutation rate for this model.
    MutationRate() float64
    // TransitionMatrix returns the conditioned mutation rate matrix
    // for this model.
    TransitionMatrix() [][]float64
    // TransitionProbs returns the conditioned transition probabilities
    // for the given state.
    TransitionProbs(char int) []float64


    // MaxPathogenPopSize returns the maximum number of pathogens allowed 
within
    // a single host of this particular host type.
    MaxPathogenPopSize() int
    // NextPathogenPopSize returns the pathogen population size for the ne
xt

    // generation of pathogens given the current population size.
    // This is used in conjunction with a population model under
    // relative fitness.
    NextPathogenPopSize(n int) int
    // ReplicationMethod returns whether relative, or absolute is used.
    ReplicationMethod() string


    // RecombinationRate returns the recombination rate for this model.
    RecombinationRate() float64


    // StatusDuration
    StatusDuration(status int) int
    ProbabilisticDuration() bool

}
```

IntrahostModel is an interface for any type of intrahost model.

type MotifModel

```
type MotifModel interface {
    // ID returns the ID for this fitness model.
    ModelID() int
    // Name returns the name for this fitness model.
    ModelName() string
    SetModelID(id int)
    SetModelName(name string)
    // ComputeFitness returns the corresponding fitness value given
    // a set of sequences as integers.
    ComputeFitness(chars ...uint8) (fitness float64, err error)


    AddMotif(sequence []uint8, pos []int, value float64) error

}
```

MotifModel is a type of FitnessModel where the fitness of a sequence depends on the presence of the particular motifs.

func EmptyMotifModel

```
func EmptyMotifModel(id int, name string) MotifModel
```

EmptyMotifModel returns a new motif model without any registered motifs.

type MutationPackage

```
type MutationPackage struct {
    // contains filtered or unexported fields

}
```

MutationPackage encapsulates information to be written to track when and where mutations occur in the simulation.

type SIRSimulation

```
type SIRSimulation struct {
    EpidemicSimulation

}
```

SIRSimulation creates and runs an SIR epidemiological simulation. Within this simulation, hosts may or may not run independent genetic evolution simulations.

func NewSIRSimulation

```
func NewSIRSimulation(config Config, logger DataLogger) (*SIRSimulation, e
rror)
```

NewSIRSimulation creates a new SI simulation.

func (*SIRSimulation) Process

```
func (sim *SIRSimulation) Process(t int)
```

Process runs the internal evolution simulation in each host. During intrahost evolution, if new mutations appear, the new sequence and ancestry is recorded to file.

func (*SIRSimulation) Run

```
func (sim *SIRSimulation) Run(i int)
```

Run instantiates, runs, and records the a new simulation.

func (*SIRSimulation) Update

```
func (sim *SIRSimulation) Update(t int)
```

Update looks at the timer or internal state to decide if the status of the host remains the same of will change. After the status updates, each host's status is recorded to file.

type SISSimulation

```
type SISSimulation struct {
    EpidemicSimulation
}
```

SISSimulation creates and runs an SIR epidemiological simulation. Within this simulation, hosts may or may not run independent genetic evolution simulations.

func NewSISSimulation

```
func NewSISSimulation(config Config, logger DataLogger) (*SISSimulation, e
rror)
```

NewSISSimulation creates a new SIS simulation.

func (*SISSimulation) Process

```
func (sim *SISSimulation) Process(t int)
```

Process runs the internal evolution simulation in each host. During intrahost evolution, if new mutations appear, the new sequence and ancestry is recorded to file.

func (*SISSimulation) Run

```
func (sim *SISSimulation) Run(i int)
```

Run instantiates, runs, and records the a new simulation.

func (*SISSimulation) Update

```
func (sim *SISSimulation) Update(t int)
```

Update looks at the timer or internal state to decide if the status of the host remains the same of will change. After the status updates, each host's status is recorded to file.

type SISimulation

```
type SISimulation struct {
    Epidemic
    DataLogger
    // contains filtered or unexported fields

}
```

SISimulation creates and runs an SI epidemiological simulation. Within this simulation, hosts may or may not run independent genetic evolution simulations.

func NewSISimulation

```
func NewSISimulation(config Config, logger DataLogger) (*SISimulation, err
or)
```

NewSISimulation creates a new SI simulation.

func (*SISimulation) Finalize

```
func (sim *SISimulation) Finalize()
```

Finalize performs processes to finish and close the simulation.

func (*SISimulation) Initialize

```
func (sim *SISimulation) Initialize(params ...interface{})
```

Initialize initializes the simulation and accepts 0 or more parameters. For example, creating datbases etc.

func (*SISimulation) InstanceID

```
func (sim *SISimulation) InstanceID() int
```

InstanceID returns the ID of the current realized simulation.

func (*SISimulation) LogFrequency

```
func (sim *SISimulation) LogFrequency() int
```

LogFrequency returns the interval in number of pathogen generation between data recordings.

func (*SISimulation) LogTransmission

```
func (sim *SISimulation) LogTransmission() bool
```

LogTransmission returns true is transmission events are saved to disk. If false, transmssion events occur but are not recorded.

func (*SISimulation) NumGenerations

```
func (sim *SISimulation) NumGenerations() int
```

NumGenerations returns the total number of pathogen generations the simulation will simulate. This is equivalent to the total number of iterations of the simulation.

func (*SISimulation) Process

```
func (sim *SISimulation) Process(t int)
```

Process runs the internal evolution simulation in each host. During intrahost evolution, if new mutations appear, the new sequence and ancestry is recorded to file.

func (*SISimulation) Run

```
func (sim *SISimulation) Run(i int)
```

Run instantiates, runs, and records the a new simulation.

func (*SISimulation) SetGenerations

```
func (sim *SISimulation) SetGenerations(n int)
```

SetGenerations sets the total number of pathogen generations the simulation will simulate. This is equivalent to the total number of iterations of the simulation.

func (*SISimulation) SetInstanceID

```
func (sim *SISimulation) SetInstanceID(i int)
```

SetInstanceID sets the instance ID of the current realized simulation.

func (*SISimulation) SetStopped

```
func (sim *SISimulation) SetStopped(b bool)
```

SetStopped sets the internal status of the current simulation. If set to true, this indicates that the simulation has stopped. If set to false, the current simulation has not yet stopped. By default, the value of internal status is false.

func (*SISimulation) SetTime

```
func (sim *SISimulation) SetTime(t int)
```

SetTime sets the current internal time of the simulation. The simulation's internal time is based on the number of iterations that has taken place. This is equivalent to the number of pathogen generations.

func (*SISimulation) Stopped

```
func (sim *SISimulation) Stopped() bool
```

Stopped returns true if the current simulation has stopped. If it returns false, the current simulation has not yet stopped

func (*SISimulation) Time

```
func (sim *SISimulation) Time() int
```

Time returns the current internal time of the simulation. The simulation's internal time should be the number of iterations that has taken place. This is equivalent to the number of pathogen generations.

func (*SISimulation) Transmit

```
func (sim *SISimulation) Transmit(t int)
```

Transmit facilitates the sampling and migration process of pathogens between hosts.

func (*SISimulation) Update

```
func (sim *SISimulation) Update(t int)
```

Update looks at the timer or internal state to decide if the status of the host remains the same of will change. After the status updates, each host's status is recorded to file.

type SQLiteLogger

```
type SQLiteLogger struct {
    // contains filtered or unexported fields

}
```

SQLiteLogger is a DataLogger that writes simulation data t0 SQLite databases. Each writer function writes to an independent SQLite database and foreign keys are added to each database at the closing phase after the simulation is completed.

func NewSQLiteLogger

```
func NewSQLiteLogger(basepath string, i int) *SQLiteLogger
```

NewSQLiteLogger creates a new logger that writes to a SQLite database.

func (*SQLiteLogger) Init

```
func (l *SQLiteLogger) Init() error
```

Init creates a new tables in the database. For example, each new realization of the simulation creates a new table for transmissions, frequencies, statuses, nodes and genotypes.

func (*SQLiteLogger) SetBasePath

```
func (l *SQLiteLogger) SetBasePath(basepath string, i int)
```

SetBasePath sets the base path of the logger.

func (*SQLiteLogger) WriteGenotypeFreq

```
func (l *SQLiteLogger) WriteGenotypeFreq(c <-chan GenotypeFreqPackage)
```

WriteGenotypeFreq records the count of unique genotype nodes present within the host in a given time in the simulation.

*func (*SQLiteLogger) WriteGenotypeNodes*

```
func (l *SQLiteLogger) WriteGenotypeNodes(c <-chan GenotypeNode)
```

WriteGenotypeNodes records new genotype node's ID and associated genotype ID to file

*func (*SQLiteLogger) WriteGenotypes*

```
func (l *SQLiteLogger) WriteGenotypes(c <-chan Genotype)
```

WriteGenotypes records a new genotype's ID and sequence to file.

*func (*SQLiteLogger) WriteMutations*

```
func (l *SQLiteLogger) WriteMutations(c <-chan MutationPackage)
```

WriteMutations records every time a new genotype node is created. It records the time and in what host this new mutation arose.

*func (*SQLiteLogger) WriteStatus*

```
func (l *SQLiteLogger) WriteStatus(c <-chan StatusPackage)
```

WriteStatus records the status of each host every generation.

*func (*SQLiteLogger) WriteTransmission*

```
func (l *SQLiteLogger) WriteTransmission(c <-chan TransmissionPackage)
```

WriteTransmission records the ID's of genotype node that are transmitted between hosts.

type SequenceNodeEpidemic

```
type SequenceNodeEpidemic struct {
    sync.RWMutex
    // contains filtered or unexported fields

}
```

SequenceNodeEpidemic is a type of Epidemic that uses a SequenceNode to represent pathogens.

func (*SequenceNodeEpidemic) DeadProcess

```
func (sim *SequenceNodeEpidemic) DeadProcess(i, t int, host Host, wg *syn
c.WaitGroup)
```

DeadProcess executes within-host processes that occurs when a host is in the dead state state that is perpetually uninfectable. This state is identically to Removed but is used to distinguish from a recovered, but perpetually immune state.

func (*SequenceNodeEpidemic) ExposedProcess

```
func (sim *SequenceNodeEpidemic) ExposedProcess(i, t int, host Host, c cha
n<- MutationPackage, wg *sync.WaitGroup)
```

ExposedProcess executes within-host processes that occurs when a host is in the exposed state. By default, it is same as InfectedProcess.

func (*SequenceNodeEpidemic) GenotypeNodeMap

```
func (sim *SequenceNodeEpidemic) GenotypeNodeMap() map[ksuid.KSUID]Genotyp
eNode
```

GenotypeNodeMap returns the set of all GenotypeNodes seen since the start of the simulation.

func (*SequenceNodeEpidemic) GenotypeSet

```
func (sim *SequenceNodeEpidemic) GenotypeSet() GenotypeSet
```

GenotypeSet returns the set of all Genotypes seen since the start of the simulation.

func (*SequenceNodeEpidemic) Host

```
func (sim *SequenceNodeEpidemic) Host(id int) Host
```

Host returns the selected host in the simulation.

func (*SequenceNodeEpidemic) HostConnection

```
func (sim *SequenceNodeEpidemic) HostConnection(a, b int) float64
```

HostConnection returns the weight of a connection between two hosts if it exists, returns 0 otherwise.

func (*SequenceNodeEpidemic) HostMap

```
func (sim *SequenceNodeEpidemic) HostMap() map[int]Host
```

HostMap returns the hosts in the simulation in the form of a map. The key is the host's ID and the value is the pointer to the host.

func (*SequenceNodeEpidemic) HostNeighbors

```
func (sim *SequenceNodeEpidemic) HostNeighbors(id int) []Host
```

HostNeighbors retrieves the directly connected hosts to the current host based on the supplied adjacency matrix.

func (*SequenceNodeEpidemic) HostStatus

```
func (sim *SequenceNodeEpidemic) HostStatus(id int) int
```

HostStatus retrieves the current status of the selected host.

func (*SequenceNodeEpidemic) HostTimer

```
func (sim *SequenceNodeEpidemic) HostTimer(id int) int
```

HostTimer returns the current number of generations remaining before the host changes status.

func (*SequenceNodeEpidemic) InfectableStatuses

```
func (sim *SequenceNodeEpidemic) InfectableStatuses() []int
```

InfectableStatuses returns the list of statuses that infected hosts can transmit to.

func (*SequenceNodeEpidemic) InfectedProcess

```
func (sim *SequenceNodeEpidemic) InfectedProcess(i, t int, host Host, c ch
an<- MutationPackage, wg *sync.WaitGroup)
```

InfectedProcess executes within-host processes that occurs when a host is in the infected state.

func (*SequenceNodeEpidemic) InfectiveProcess

```
func (sim *SequenceNodeEpidemic) InfectiveProcess(i, t int, host Host, c c
han<- MutationPackage, wg *sync.WaitGroup)
```

InfectiveProcess executes within-host processes that occurs when a host is in the infective state. By default, it is same as InfectedProcess.

func (*SequenceNodeEpidemic) NewInstance

```
func (sim *SequenceNodeEpidemic) NewInstance() (Epidemic, error)
```

NewInstance creates a new instance from the stored configuration

func (*SequenceNodeEpidemic) RecoveredProcess

```
func (sim *SequenceNodeEpidemic) RecoveredProcess(i, t int, host Host, wg
*sync.WaitGroup)
```

RecoveredProcess executes within-host processes that occurs when a host is in the recovered state that is perpetually uninfectable. This state is identically to Removed but is used to distinguish from a dead state.

func (*SequenceNodeEpidemic) RemovedProcess

```
func (sim *SequenceNodeEpidemic) RemovedProcess(i, t int, host Host, wg *s
ync.WaitGroup)
```

RemovedProcess executes within-host processes that occurs when a host is in the removed state that is perpetually uninfectable.

func (*SequenceNodeEpidemic) SetHostStatus

```
func (sim *SequenceNodeEpidemic) SetHostStatus(id, status int)
```

SetHostStatus sets the current status of the selected host to a given status code.

func (*SequenceNodeEpidemic) SetHostTimer

```
func (sim *SequenceNodeEpidemic) SetHostTimer(id, interval int)
```

SetHostTimer sets the number of generations for the host to remain in its current status.

func (*SequenceNodeEpidemic) StopSimulation

```
func (sim *SequenceNodeEpidemic) StopSimulation() bool
```

StopSimulation check whether the simulation has satisfied at least one of the conditions that will halt the simulation in the current interation. Returns true is the simulation should stop, false otherwise.

func (*SequenceNodeEpidemic) SusceptibleProcess

```
func (sim *SequenceNodeEpidemic) SusceptibleProcess(i, t int, host Host, w
g *sync.WaitGroup)
```

SusceptibleProcess executes within-host processes that occurs when a host is in the susceptible state.

func (*SequenceNodeEpidemic) VaccinatedProcess

```
func (sim *SequenceNodeEpidemic) VaccinatedProcess(i, t int, host Host, wg
 *sync.WaitGroup)
```

VaccinatedProcess executes within-host processes that occurs when a host is in a globally immune state with the chance to become globally susceptible again.

type StatusPackage

```
type StatusPackage struct {
    // contains filtered or unexported fields
}
```

StatusPackage encapsulates the data to be written everytime the status of a host has to be recorded.

type StopCondition

```
type StopCondition interface {
    Reason() string
    Check(sim Epidemic) bool
}
```

StopCondition describes simulation conditions that must be satisfied in order for the simulation to continue. The Check method checks if the simulation still satisfies the imposed condition.

func NewAlleleExistsCondition

```
func NewAlleleExistsCondition(char uint8, site int) StopCondition
```

NewAlleleExistsCondition creates a new StopCondition that stops the simulation once the given char at a particular site becomes extinct.

func NewAlleleFixedLostCondition

```
func NewAlleleFixedLostCondition(char uint8, site int) StopCondition
```

NewAlleleFixedLostCondition creates a new StopCondition that stops the simulation once the particular allele in a given site has either been fixed or been lost.

func NewGenotypeExistsCondition

```
func NewGenotypeExistsCondition(sequence []uint8) StopCondition
```

NewGenotypeExistsCondition creates a new StopCondition that stops the simulation once the given sequence genotype becomes extinct.

type TransmissionEvent

```
type TransmissionEvent struct {
    // contains filtered or unexported fields
}
```

TransmissionEvent is a struct for sending and receiving transmission event information.

type TransmissionModel

```
type TransmissionModel interface {
    // ID returns the ID for this transmission model.
    ModelID() int
    // Name returns the name for this transmission model.
    ModelName() string
    SetModelID(id int)
    SetModelName(name string)
    // TransmissionProb returns the probability that a transmission event
    // occurs between one host and one neighbor (per capita event) occurs.
    TransmissionProb() float64

    // TransmissionSize returns the number of pathogens transmitted given
    // a transmission event occurs.
    TransmissionSize() int
}
```

TransmissionModel describes the transmission probability and number of pathogens that transmits per event. The model may be constant or probabilistic.

type TransmissionPackage

```
type TransmissionPackage struct {
    // contains filtered or unexported fields
}
```

TransmissionPackage encapsulates information to be written to track the movement of genotype nodes across the host population.

APPENDIX III
SIMULATION PARAMETERS

FIXATION PROBABILITY OF A MUTATION SPREADING OVER A NETWORK

Table 12. Parameters used to test the effect of network topology on fixation probability of mutations.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	10000
Number of independent realizations	num_instances	1000
Number of hosts	host_popsize	20, 200
Epidemic model	epidemic_model	"sis"
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	["0", "1"]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	"constant"
Constant population size	constant_pop_size	500
Duration of infection	infected_duration	10
Fitness model	fitness_model	"multiplicative"
Transmission mode	transmission_mode	"poisson"
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	5.0
Condition that halts the simulation	condition	"allele_fixloss"
Target allele	sequence	"1"
Position of target allele	position	0

Table 13. Parameters used to simulate the well-mixed population null model for testing the effect of network topology.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	10000
Number of independent realizations	num_instances	1000
Number of hosts	host_popsize	1
Epidemic model	epidemic_model	“endtrans”
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“constant”
Constant population size	constant_pop_size	500, 900
Duration of infection	infected_duration	10
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	0.0
Transmission size	transmission_size	0.0
Condition that halts the simulation	condition	“allele_fixloss”
Target allele	sequence	“1”
Position of target allele	position	0

Table 14. Parameters used to test the effect of network topology on the genetic diversity of pathogen spreading over a network.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	1000
Number of independent realizations	num_instances	100
Number of hosts	host_popsize	20, 200
Epidemic model	epidemic_model	"sis"
Coinfection	coinfection	true, false
Number of sites	num_sites	1000
Expected characters	expected_characters	["0", "1"]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	1e-05
Transition rate matrix	transition_matrix	[[0.0e+00, 1.0e+00], [1.0e+00, 0.0e+00]]
Recombination rate	recombination_rate	0.0
Replication model	replication_model	"constant"
Constant population size	constant_pop_size	500
Duration of infection	infected_duration	10
Fitness model	fitness_model	"multiplicative"
Transmission mode	transmission_mode	"poisson"
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	5.0

Table 15. Parameters used to simulate the well-mixed population null model for testing the effect of network topology on genetic diversity.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	1000
Number of independent realizations	num_instances	100
Number of hosts	host_popsize	1
Epidemic model	epidemic_model	“si”
Coinfection	coinfection	false
Number of sites	num_sites	1000
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	1e-05
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“bht”
Growth rate	growth_rate	10.0
Maximum population size	max_pop_size	90000
Duration of infection	infected_duration	1001
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	0.0
Transmission size	transmission_size	0.0

FIXATION PROBABILITY OF A MUTATION OVER SUCCESSIVE TRANSMISSIONS

Table 16. Parameters used to test the effect of transmission bottlenecks on fixation probability of mutations.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	10000
Number of independent realizations	num_instances	1000, 2000, 5000
Number of hosts	host_popsize	2001
Epidemic model	epidemic_model	“endtrans”
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“constant”
Constant population size	constant_pop_size	500
Duration of infection	infected_duration	10, 20
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	5.0
Condition that halts the simulation	condition	“allele_fixloss”
Target allele	sequence	“1”
Position of target allele	position	0

Table 17. Parameters used to simulate the well-mixed population null model for testing the effect of transmission bottlenecks on fixation probability.

Parameter name	**Keyword**	**Value/s used**
Number of generations	num_generations	10000
Number of independent realizations	num_instances	1000, 2000, 5000
Number of hosts	host_popsize	1
Epidemic model	epidemic_model	"endtrans"
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	["0", "1"]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	"constant"
Constant population size	constant_pop_size	10, 50, 500
Duration of infection	infected_duration	10, 20
Fitness model	fitness_model	"multiplicative"
Transmission mode	transmission_mode	"constant"
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	10, 50, 500
Condition that halts the simulation	condition	"allele_fixloss"
Target allele	sequence	"1"
Position of target allele	position	0

Table 18. Parameters used to simulate harmonic mean population model for testing the effect of transmission bottlenecks on fixation probability.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	10000
Number of independent realizations	num_instances	1000, 2000, 5000
Number of hosts	host_popsize	1
Epidemic model	epidemic_model	“endtrans”
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“constant”
Constant population size	constant_pop_size	46, 84, 264, 344
Duration of infection	infected_duration	10, 20
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	46, 84, 264, 344
Condition that halts the simulation	condition	“allele_fixloss”
Target allele	sequence	“1”
Position of target allele	position	0

PROPORTION OF SUCCESSFUL FIXATIONS OF A MUTATION IN AN EXPANDING TRANSMISSION TREE

Table 19. Parameters used to test the effect of transmission tree topology on fixation probability of mutations.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	1000
Number of independent realizations	num_instances	100
Number of hosts	host_popsize	11, 14, 15, 18, 25
Epidemic model	epidemic_model	“endtrans”
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“constant”
Constant population size	constant_pop_size	1000
Duration of infection	infected_duration	250
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	10

Table 20. Parameters used to test the effect of transmission path topology on fixation probability of mutations.

Parameter name	Keyword	Value/s used
Number of generations	num_generations	750, 1000
Number of independent realizations	num_instances	100
Number of hosts	host_popsize	3, 4, 5, 6
Epidemic model	epidemic_model	“endtrans”
Coinfection	coinfection	false
Number of sites	num_sites	1
Expected characters	expected_characters	[“0”, “1”]
Logging frequency	log_freq	1
Mutation rate	mutation_rate	0.0
Recombination rate	recombination_rate	0.0
Replication model	replication_model	“constant”
Constant population size	constant_pop_size	1000
Duration of infection	infected_duration	250
Fitness model	fitness_model	“multiplicative”
Transmission mode	transmission_mode	“constant”
Transmission probability	transmission_prob	1.0
Transmission size	transmission_size	10

APPENDIX IV

SUPPLEMENTAL TRANSMISSION CHAIN MODELS

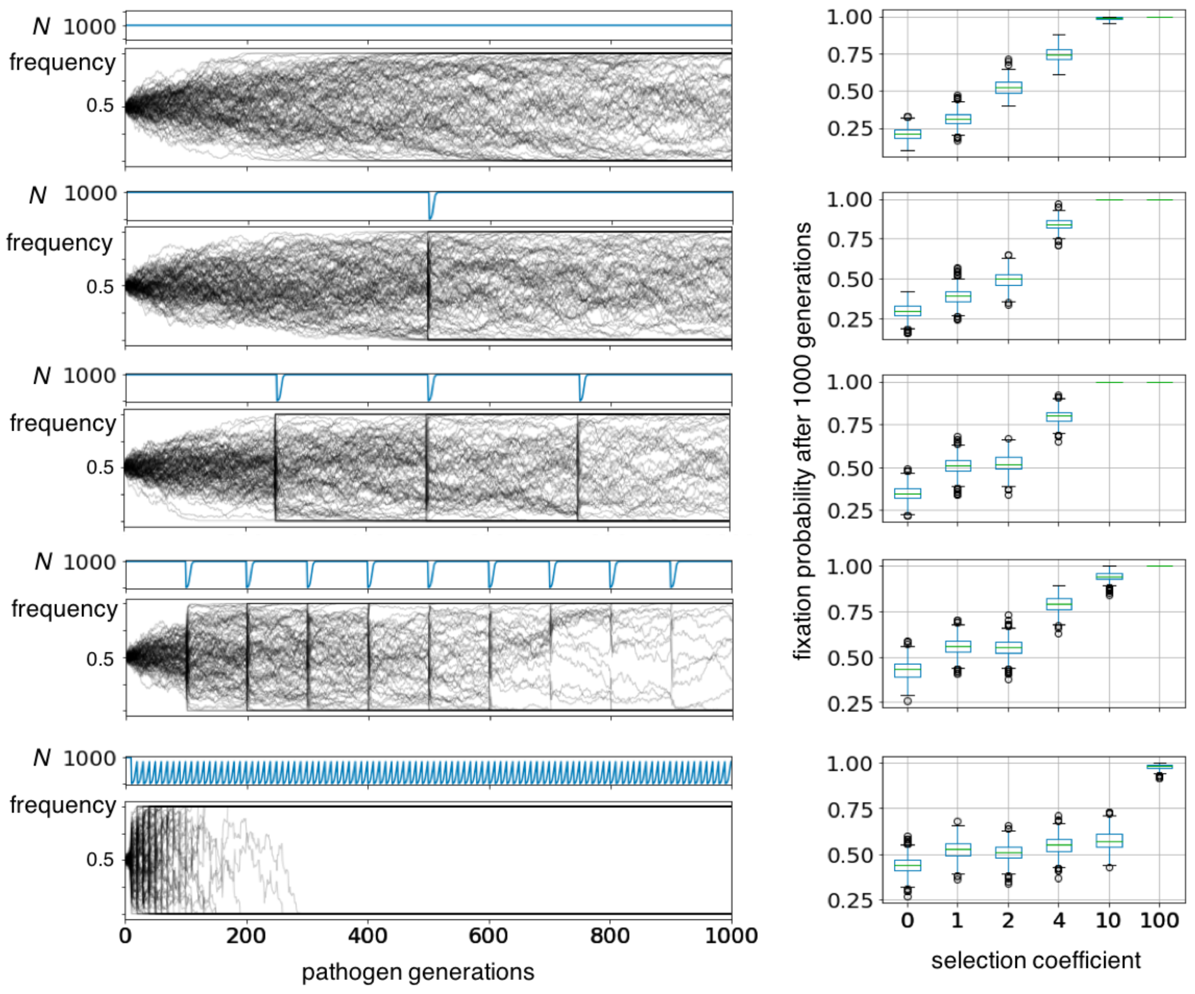


Figure 38. Effect of frequency of transmission bottlenecks on the fixation probability of an allele after 1000 pathogen generations.

Graphs on the left indicate the frequency of population bottlenecks (blue line) and the change in frequency of the preferred allele starting (black line). Graphs on the right indicate the conditional fixation probability for a range of scaled selection coefficients Ns after 1000 pathogen generations.

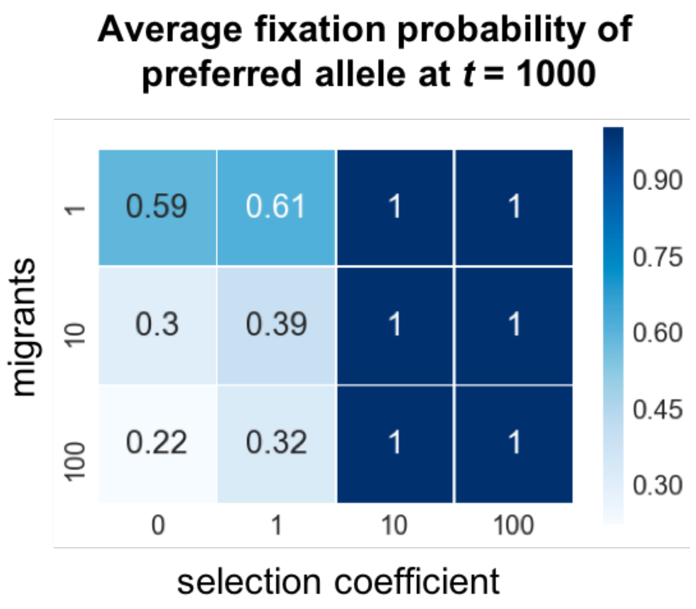


Figure 39. Effect of the number of pathogens transmitted and selection coefficient on the fixation probability of an allele after 1000 pathogen generations.

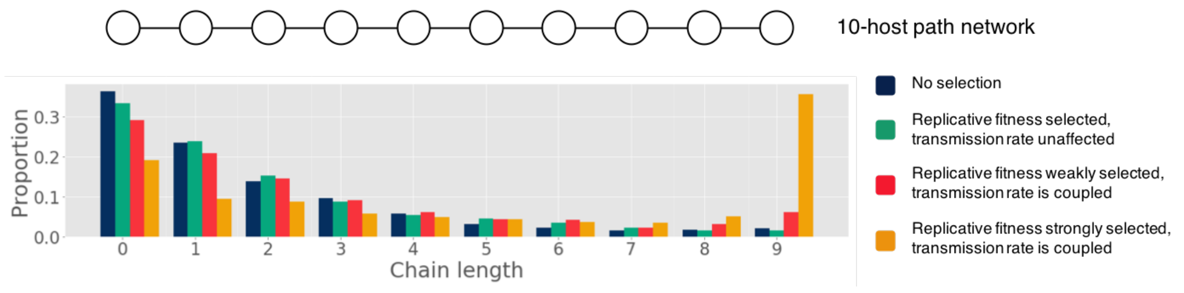


Figure 40. Transmission chain length frequency with or without selection on the transmissibility of the pathogen.

Height of the bar indicates the proportion of times the transmission chain achieves a certain length. For example, a chain length of zero means that no transmission occurred while a chain length of one means one transmission happened. The top figure shows the chain topology of the network being simulated.

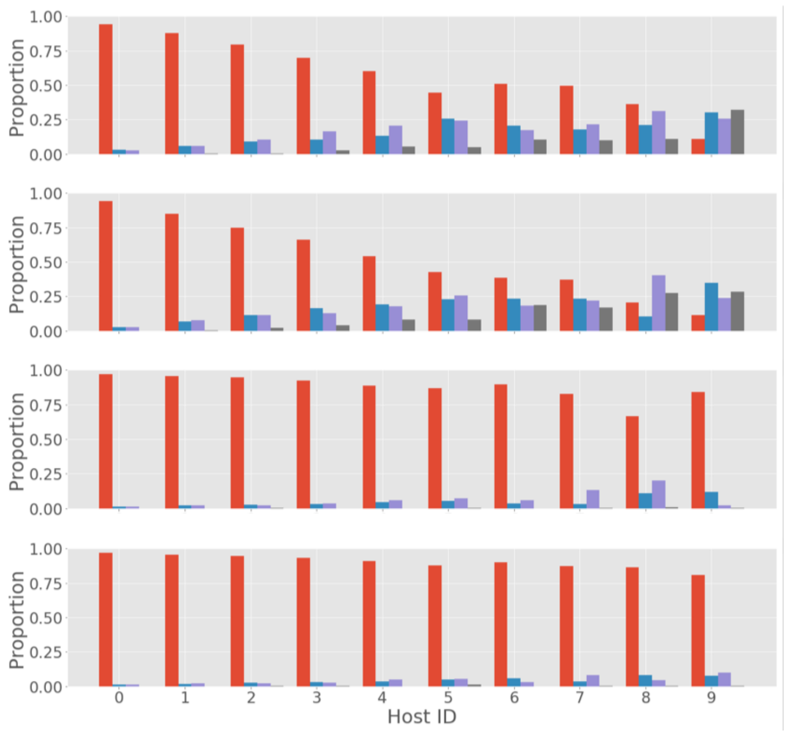


Figure 41. Frequency of genotypes under neutral evolution, replicative selection, and transmission selection.

Height of the bar indicates the proportion of times the transmission chain achieves a certain length. For example, a chain length of zero means that no transmission occurred while a chain length of one means one transmission happened.

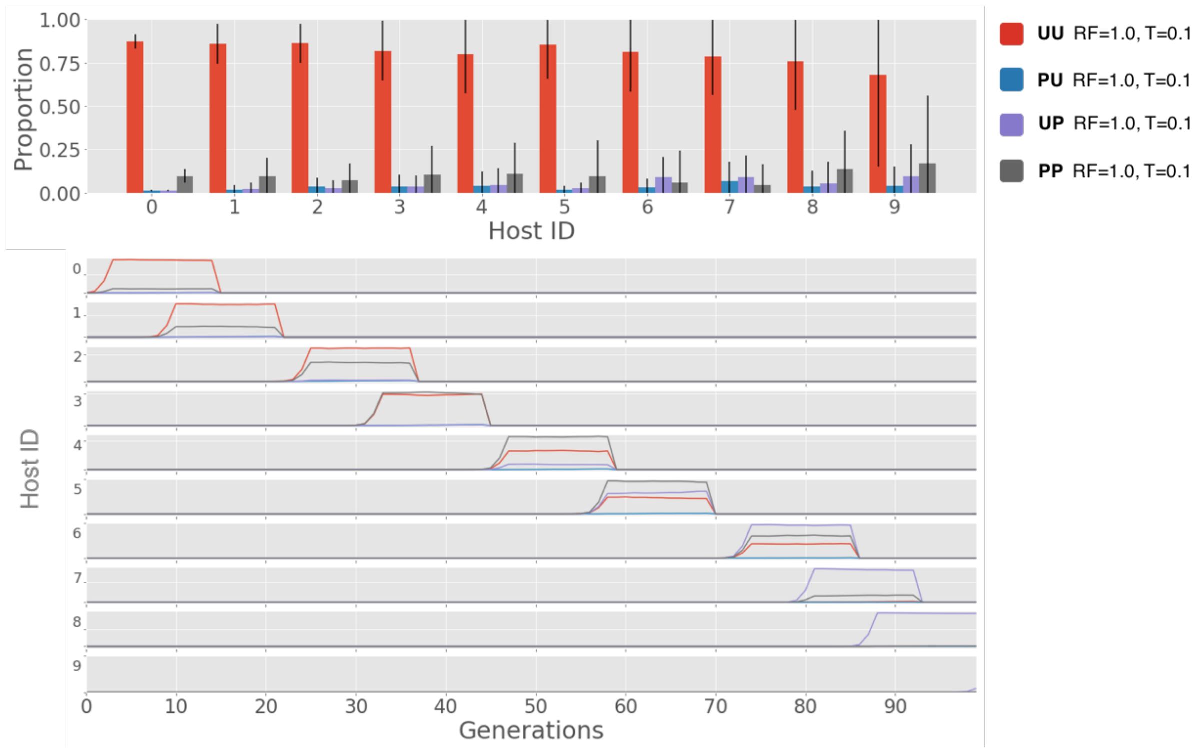


Figure 42. Frequency of genotypes under neutral evolution across the transmission chain.

Top graph: height of the bar indicates the proportion of times the transmission chain achieves a certain length. For example, a chain length of zero means that no transmission occurred while a chain length of one means one transmission happened. Bottom set of graphs indicate intrahost allele frequency.

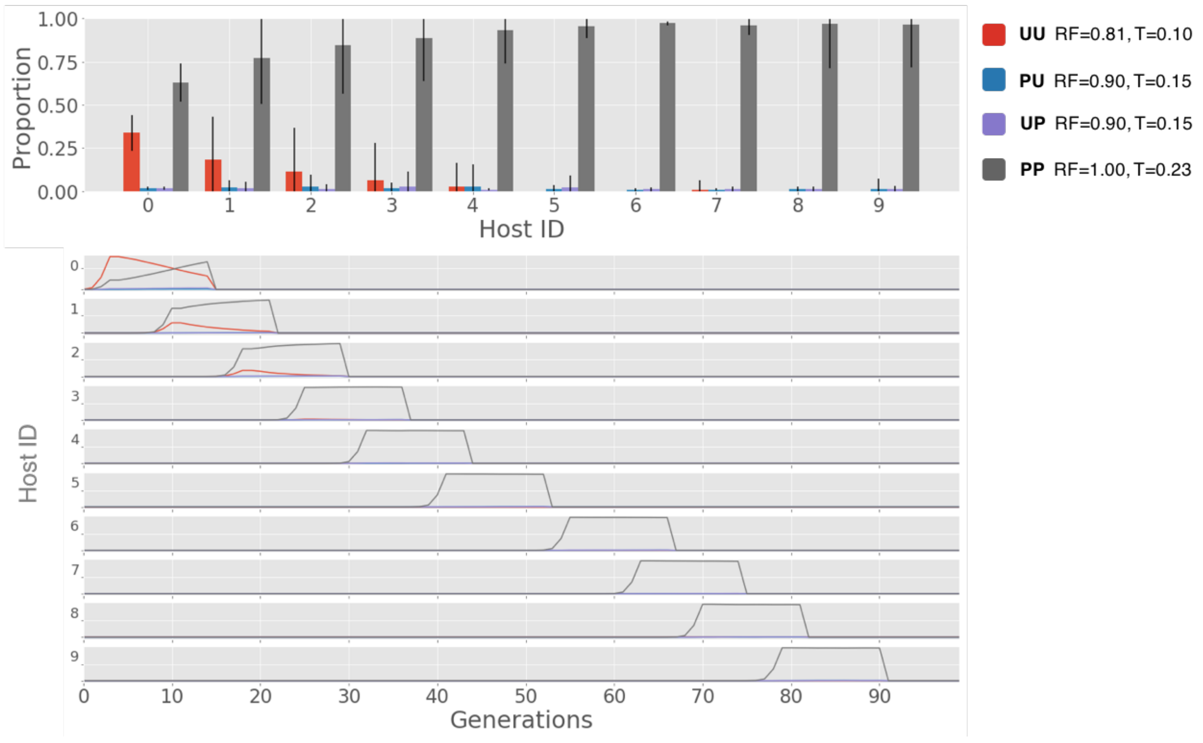


Figure 43. Frequency of genotypes under transmission selection across the transmission chain.

Top graph: height of the bar indicates the proportion of times the transmission chain achieves a certain length. For example, a chain length of zero means that no transmission occurred while a chain length of one means one transmission happened. Bottom set of graphs indicate intrahost allele frequency.

APPENDIX V

SUPPLEMENTAL TRANSMISSION TREE MODELS

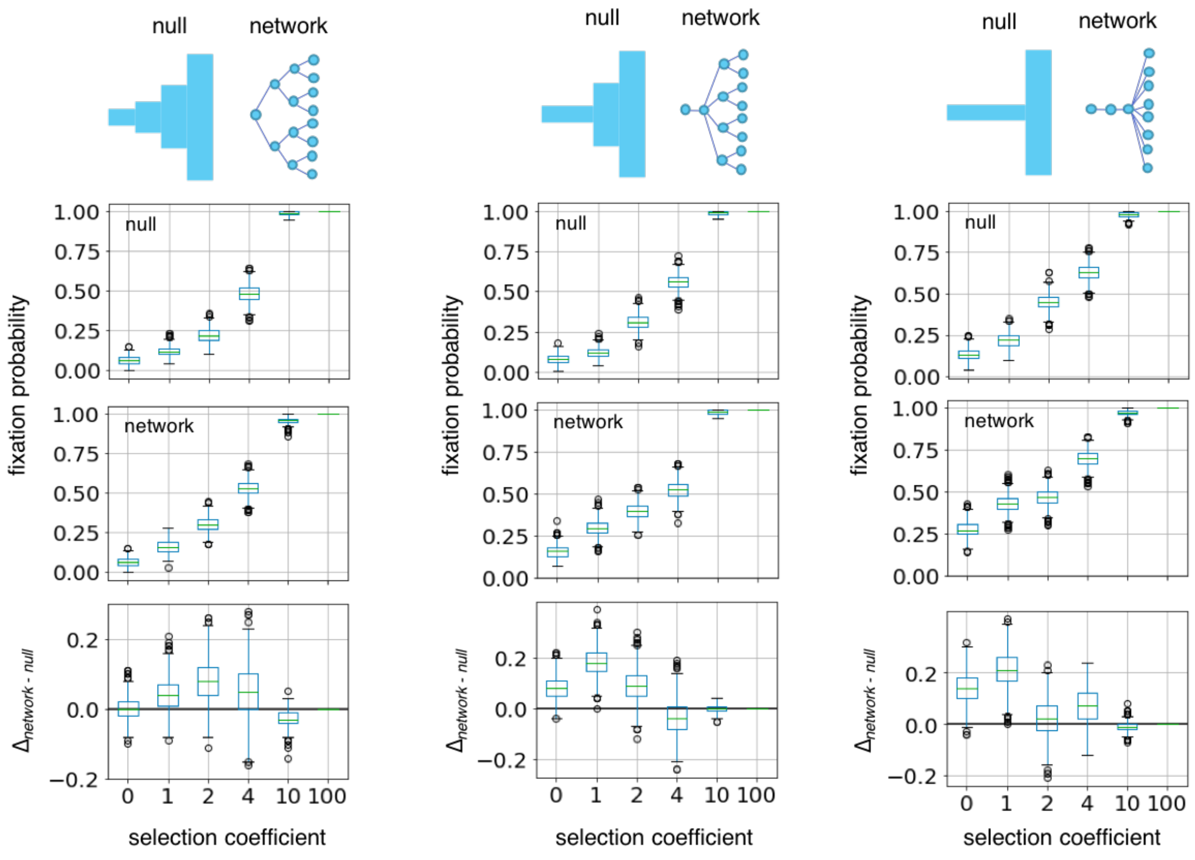


Figure 44. Fixation probability after 1000 generations for regular, heterogenous, and superspreader transmission trees.

Top row shows the size of the total intrahost population and the network topology for each transmission tree. Second and third rows show the fixation probability after 1000 pathogen generation under the unstructured null model and the network model respectively. The fourth row shows the difference in fixation probabilities per network type.

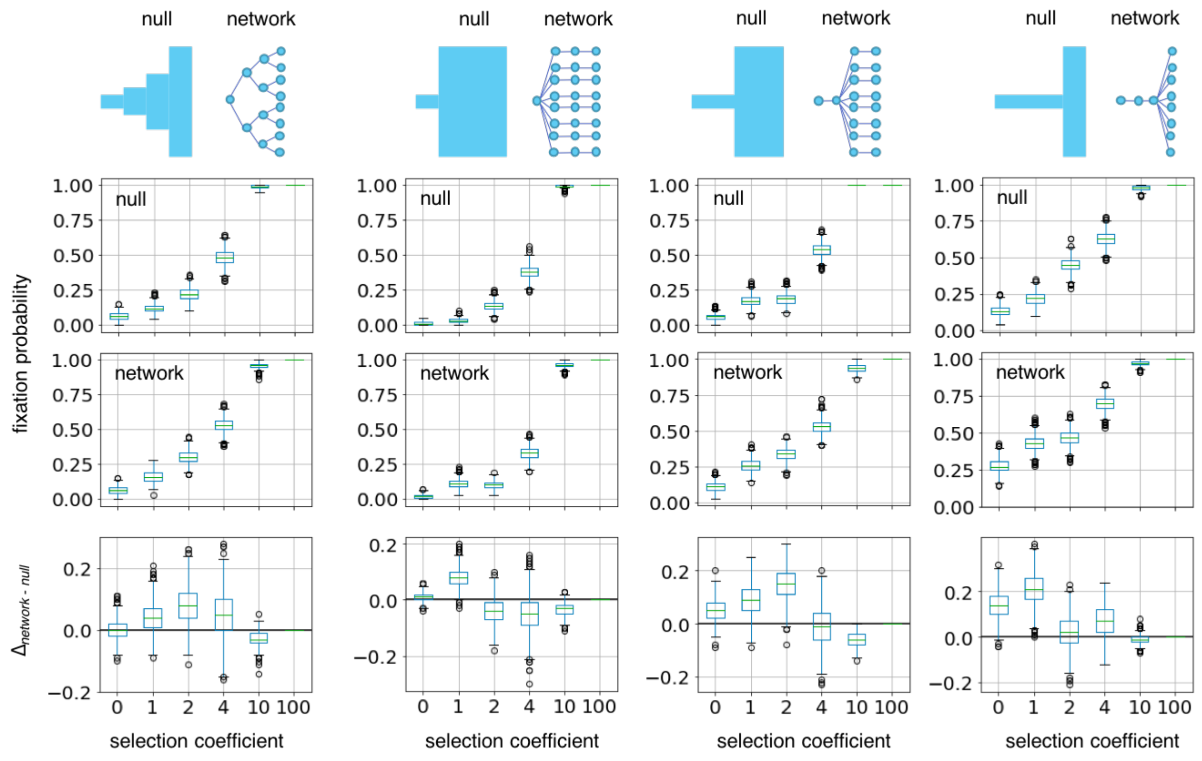


Figure 45. Fixation probability after 1000 generations for regular, early-spreading, late-spreading, and superspreader transmission trees.

Top row shows the size of the total intrahost population and the network topology for each transmission tree. Second and third rows show the fixation probability after 1000 pathogen generation under the unstructured null model and the network model respectively. The fourth row shows the difference in fixation probabilities per network type.

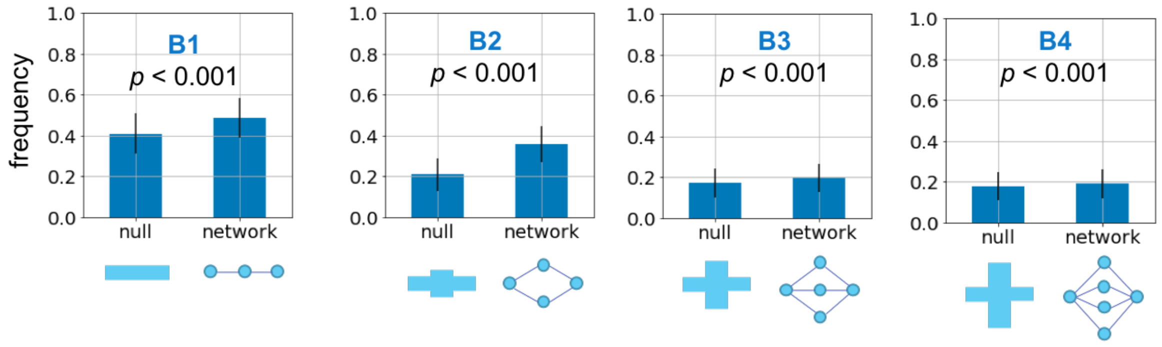


Figure 46. Fixation probability after 1000 generations for linear and diverging transmission paths.

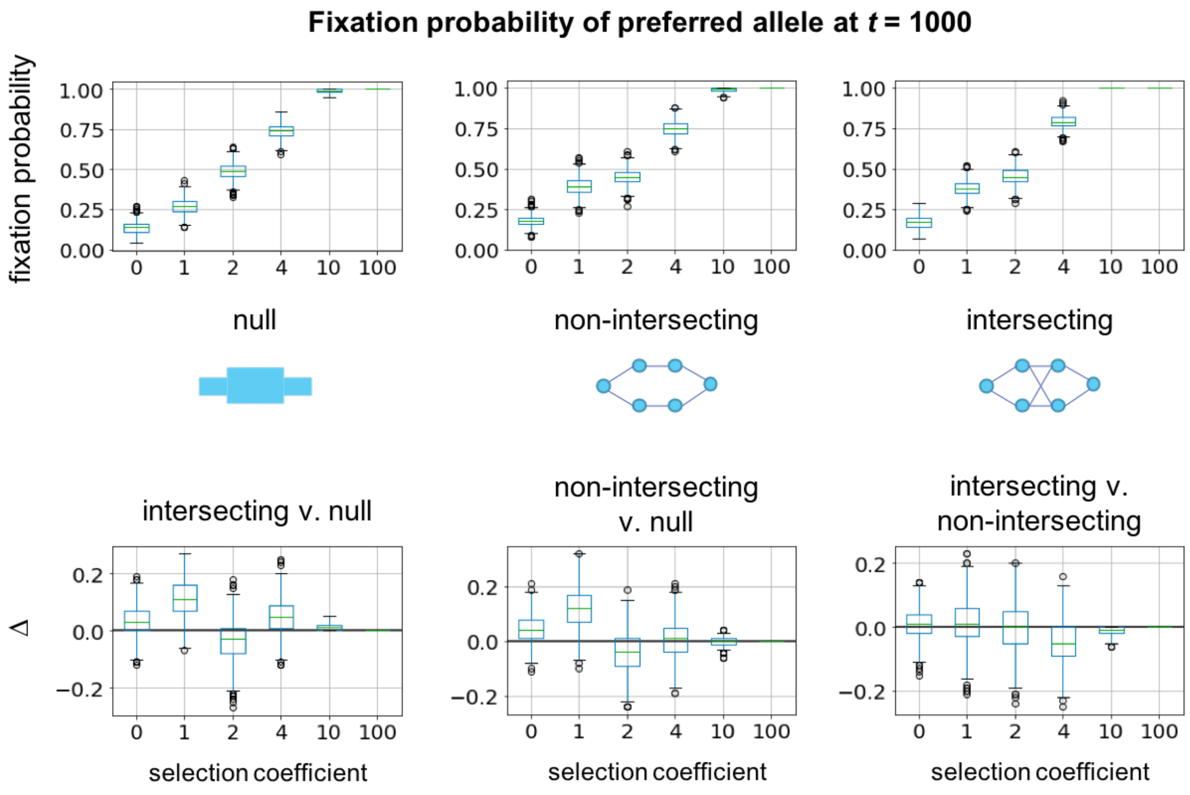


Figure 47. Fixation probability after 1000 generations for diverging transmission paths.

Top shows the average fixation probability after 1000 generations for different diverging network structures under neutral evolution and different scaled selection coefficients. Bottom shows the difference between fixation probabilities after 1000 generations of the unstructured null model and the network model.