A Stochastic Simulator of Birth–Death Master Equations with Application to Phylodynamics

Timothy G. Vaughan*[,1,2] and Alexei J. Drummond[1,3]

[1]Allan Wilson Centre for Molecular Ecology and Evolution, Massey University, Palmerston North, New Zealand

[2]Institute of Veterinary Animal and Biomedical Sciences, Massey University, Palmerston North, New Zealand

[3]Department of Computer Science, University of Auckland, Auckland, New Zealand

***Corresponding author:** E-mail: t.g.vaughan@massey.ac.nz.

Associate editor: Asger Hobolth

Abstract

In this article, we present a versatile new software tool for the simulation and analysis of stochastic models of population phylodynamics and chemical kinetics. Models are specified via an expressive and human-readable XML format and can be used as the basis for generating either single population histories or large ensembles of such histories. Importantly, phylogenetic trees or networks can be generated alongside the histories they correspond to, enabling investigations into the interplay between genealogies and population dynamics. Summary statistics such as means and variances can be recorded in place of the full ensemble, allowing for a reduction in the amount of memory used—an important consideration for models including large numbers of individual subpopulations or demes. In the case of population size histories, the resulting simulation output is written to disk in the flexible JSON format, which is easily read into numerical analysis environments such as R for visualization or further processing. Simulated phylogenetic trees can be recorded using the standard Newick or NEXUS formats, with extensions to these formats used for non-tree-like inheritance relationships.

Key words: **stochastic simulation, population genetics, phylogenetic trees, chemical kinetics simulation, epidemic modeling.**

Article

Introduction

Regardless of their composition, biological populations are subject to a huge variety of influences and complexities that are not easily quantified or incorporated into a mathematical analysis. It is for this reason that the most realistic and generally applicable models are often stochastic in nature, and cannot generally be analyzed without simulation. Combined with the increasing prevalence of genetic data sampled from measurably evolving populations (Drummond et al. 2003), particularly in the context of pathogen evolution (Pybus and Rambaut 2009; Kühnert et al. 2011), there is a clear need for accessible software capable of simultaneously analyzing the demographic and genealogical predictions under general stochastic models of population dynamics.

As such models abound, it is unsurprising that a correspondingly large variety of stochastic simulators also exists. In the context of population genetics, recent examples include Nemo (Guillaume and Rougemont 2006) and FFPopSim (Zanini and Neher 2012), both of which perform simulations under models which assume discrete generation times and allow for direct simulation of sequence evolution. Nemo is an application and flexible programming framework suitable for performing such simulations and is agent-based in design, meaning that it is capable of directly simulating complex life cycles that correspond to non-Markovian dynamics at the population level. In contrast, FFPopSim is a collection of library routines scriptable in the Python programming language that can perform similar simulations using either an agent-based approach or by grouping individuals into genetically homogeneous subpopulations. Although the latter approach restricts the number of loci that can be treated, it allows for the modeling of larger populations evolving under sophisticated fitness landscapes.

Although useful for studying traditional population genetics models, neither of these programs allow for genealogy simulation. Another population genetics simulator, TreesimJ (O'Fallon 2010), allows for joint simulation of the population history, genetic sequences and the genealogy of the individuals present at each (discrete) generation, discarding extinct lineages along the way. This clever technique enables generation of reasonably large genealogies conditional on the final population. However, TreesimJ only permits variation in the population size itself under a fixed number of hard-coded deterministic models.

The connection between models of population dynamics and those of chemical kinetics has a long history; indeed, the Lotka–Volterra equations for predator–prey dynamics (Volterra 1926) were originally developed to model a set of self-catalyzing chemical reactions (Lotka 1920). As such, it is important to be aware of the wide variety of stochastic chemical kinetics simulators which exist. An important example is Dizzy (Ramsey et al. 2005), which is a sophisticated reaction modeling system allowing the simulation of the progress of arbitrary reaction systems using a variety of stochastic (and

deterministic) simulation algorithms. Models are specified using either the polished graphical interface or can be imported (and exported) using either the systems biology markup language (SBML) format (Hucka et al. 2003) or a custom human-readable text format. The package StockKit2 (Sanft et al. 2011) also allows specification using SBML, although there the emphasis is on the efficiency of the simulation engines themselves.

In contrast to the population genetics simulators, the chemical kinetics software above deals with continuous-time models where individual "demographic" events occur at times separated by exponentially distributed intervals. This is a more appropriate model of real population dynamics for settings in which demographic events are not constrained to occur at discrete times. However, due to their focus on chemical kinetics, these software packages are not capable of simulating genealogies under their reaction systems and thus do not meet our requirement.

In general, none of the software packages described above allow for both flexible simulation of arbitrary stochastic processes and the simultaneous generation of genealogies under these processes. We have therefore developed a new package to fill this lacuna. Named MASTER (moments and stochastic trees from event reactions), this tool takes descriptions of continuous time stochastic chemical-kinetics style models expressed in a succinct and human-readable XML format as its input, generates one or more simulated population size histories and genealogies, then writes the results to portable output files for easy visualization or further analysis using third party tools such as R (R Core Team 2012) and FigTree (Rambaut 2012). By default, simulations are performed using the exact direct method (Gillespie 1976), although alternative approximate methods can be employed when simultaneous genealogy generation is not required, leading to a significant performance boost in these cases. Simulations can be set up to terminate at a specific time or when some condition is met. Being primarily a population rather than agent-based simulator, MASTER does not currently simulate sequence evolution. However, MASTER has extensive support for structured populations, which can be used to represent allele frequencies, individual genotypes with small numbers of loci or lumped regions of sequence space as in the error class model of Swetina and Schuster (1982).

Note that in contrast to TreesimJ, MASTER typically simulates genealogies forward in time rather than maintaining genealogies conditional on surviving lineages. On the one hand, this allows us to generate a wider variety of genealogies containing even those lineages which have become extinct. On the other hand, it means that MASTER's genealogies tend to grow continually as the simulation progresses and may eventually become unwieldy. Partly to combat this, MASTER allows one to track the genealogy of a subset of the population present at the start of the simulation, making genealogy simulation feasible even when extremely large populations are involved. This is also desirable when simulating the genealogy of the population descendent from an invader, or when modeling pathogen transmission trees where it is only the genealogy of an infected subpopulation that is of interest.

At its heart, MASTER is an "add-on" to version 2 of the popular phylogenetics software package BEAST (Drummond and Rambaut 2007; Drummond et al. 2012), although experience with BEAST is not required for its use. Interested readers should refer to the project's web page, which is located at http://tgvaughan.github.com/MASTER/ (last accessed March 29, 2013), where the software, installation instructions and links to additional documentation and tutorials can be found. The software is distributed under the GNU General Public License version 3.

Results

MASTER can perform a variety of simulations under models having the form presented in later section. Each simulation falls into one the following generic "simulation types":

- Trajectory: This encompasses simulations that generate a single history (i.e., a stochastic realization) of the size of populations present in the model.
- Ensemble: This describes calculations that produce several independent population size trajectories.
- Ensemble Summary: This is another type of simulation that involves multiple population size trajectories, but in this case the results are summarized on-the-fly in terms of time-dependent summary statistics or moment estimates.
- Inheritance Trajectory: This encompasses joint simulations of the dynamics of population sizes and a tree or network describing the inheritance relationships between a subset of individuals within those populations.
- Inheritance Ensemble: This allows for the production of several independent inheritance trajectory simulations.

In this section, we explore the use of these simulation types, demonstrating their application to the treatment of stochastic population dynamical models of increasing complexity (a more complete description of the syntax of the XML input files is provided in later section). Additionally, we present a brief analysis of the computational efficiency of the software.

A Simple Epidemiological Model

We begin by demonstrating the application of MASTER to the simulation of a basic but fundamental model of epidemic dynamics. What is now known among mathematical epidemiologists as the SIR model is a special case of a formalism introduced nearly a century ago by Kermack and McKendrick (1927). It is so-named because it involves dividing individuals in an otherwise homogeneous population into three classes or "compartments" named *S*, *I*, and *R*. Members of compartment *S* are considered susceptible to infection by individuals from compartment *I*, which contains infected and (importantly) infectious individuals. Compartment *R* contains those individuals for whom the infection has run its course and can therefore be considered recovered (although removed might

```
<beast version='2.0' namespace='beast.core.parameter:master.beast'>

 <run spec='Trajectory' simulationTime='50'>

  <model spec='Model'>

   <!-- Compartment populations in model-->
   <population spec='Population' populationName='S' id='S'/>
   <population spec='Population' populationName='I' id='I'/>
   <population spec='Population' populationName='R' id='R'/>

   <!-- Reactions giving rise to stochastic dynamics -->
   <reaction spec='Reaction' reactionName='Infection' rate='0.001'>
     S + I -> 2I
   </reaction>
   <reaction spec='Reaction' reactionName='Recovery' rate='0.2'>
     I -> R
   </reaction>

  </model>

  <!-- Initial compartment occupancies -->
  <initialState spec='InitState'>
   <populationSize spec='PopulationSize' population='@S' size='999'/>
   <populationSize spec='PopulationSize' population='@I' size='1'/>
  </initialState>

  <!-- Output file specification -->
  <output spec='JsonOutput' fileName='SIR_output.json'/>

 </run>
</beast>
```

Fig. 1. MASTER input file specifying a single fixed time length simulation of a stochastic SIR model.

be a better description, as the model applies equally well to lethal pathogens).

Although originally presented as a deterministic model in which the population sizes in each compartment are described by continuous variables evolving under a system of ordinary differential equations, stochastic extensions have been studied for almost as long as the deterministic model (see Britton 2010 for a recent survey). Employing the chemical reaction formalism discussed in later section, we can express such a model in the following way:

$$S + I \xrightarrow{\beta} 2I \tag{1}$$

$$I \xrightarrow{\gamma} R \tag{2}$$

The first reaction describes the infection of a single susceptible individual by an infectious neighbor at average rate β (per susceptible individual per infectious individual), whereas the second describes a recovery event at average rate γ (per infectious individual). Together, these two reactions completely determine a master equation for continuous-time evolution of the probability distribution $P(N_S(t), N_I(t), N_R(t))$ with $N_i(t)$ being the population size of compartment i at time t.

The basic MASTER input file, which can be used to simulate the population size dynamics under this model, is shown in figure 1. Here, we have selected the "Trajectory" simulation type and have used the `simulationTime` attribute to specify that the simulation should run for 50 time units. The `<model>` element has been used to specify that the model contains the three compartment populations, and that these populations are subject to the two reactions given earlier. The `<initialState>` element has been used to initialize the simulation with 1,000 individuals, including 999 susceptible and 1 infected (note that we do not need to explicitly set the R compartment population size to zero, as this is the default). Finally, the `<output>` element is used to specify a JSON-formatted output file named "SIR_output.json."

To actually run the simulation, the file containing this XML must be provided to MASTER. Exactly how this is done depends on the details of the installation and the user's operating system, and is described on the project's website. Once complete, we can read the resulting output file into an active R session using the `fromJSON` function of the `rjson` library (Couture-Beil 2013) in the following way:

```
> library(rjson)
> df ← fromJSON(file= 'SIR_output.json')
```

The `df` variable then contains the simulation results including the times at which the simulated infection/recovery events occurred and the sizes of the compartment populations immediately following those events. This can be used to create plots of the simulated population dynamics. For instance, a graph of the infected population dynamics can be produced using:

```
> plot(df$t, df$I, 's')
```

Figure 2 displays each of the simulated compartment population size histories visualized in this way.

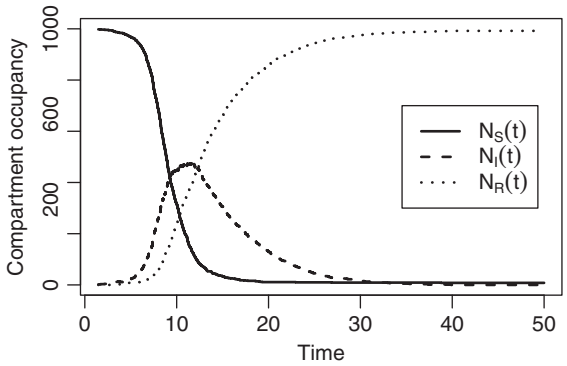

FIG. 2. Visualization of a typical SIR simulation result, using R in combination with the `rjson` library.

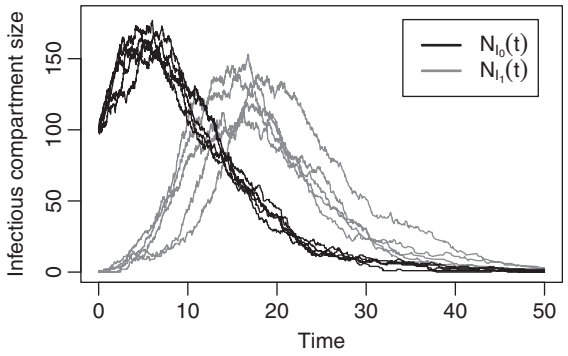

FIG. 3. Histories generated using the two-deme structured SIR model. Note the clear delay between peak infection in deme 0 and peak infection in deme 1.

Modeling Epidemics in Structured Populations

An obvious extension to the basic SIR model is one in which the community in which the epidemic is taking place is structured in some way. This occurs naturally in real populations due to geography-imposed segregation, where localities such as cities can be considered approximately homogeneous but where the relatively slower migration between cities prevents this approximation from holding in the meta-population.

We simulate this situation by considering the following simple extension to reactions of the previous model:

$$S_i + I_i \xrightarrow{\beta} 2I_i \tag{3}$$

$$I_i \xrightarrow{\gamma} R \tag{4}$$

$$S_i \xrightarrow{M^S_{ij}} S_j \tag{5}$$

$$I_i \xrightarrow{M^I_{ij}} I_j \tag{6}$$

Here, the susceptible and infected compartments S and I have been subdivided into compartments indexed by i, with different values of i representing different geographical locations or demes. The revised infection reaction limits contact to infectious and susceptible individuals belonging to compartments having the same location. The two additional reactions account for movement of infectious and susceptible individuals between locations. Note that there is no need to subdivide the recovered compartment R, as these individuals do not affect the progress of the dynamics.

The MASTER script listed in Appendix (see Structured Epidemiological Model) uses the "Ensemble" simulation type to generate a set of five independent trajectories from a two-location version of this model. The number of trajectories is specified using the `nTraj` attribute of the `<run>` element. The model specification is similar to that of the unstructured SIR model, but there are three important differences. First, `<populationType>` elements are used in place of `<population>` for the S and I compartments, allowing the nonunity number of locations to be specified. Second, the reaction elements have been grouped using named `<reactionGroup>` elements, each containing similar reactions repeated across different locations. Finally, the reaction specifications themselves specifically include population locations for the S and I types. Note that this input file specifies symmetric migration rates between the two locations, and infected individuals are just as likely to migrate as their healthy counterparts.

The use of internally structured population types also results in a slightly different specification of the initial state: full `<population>` elements of `spec` value `Population` are needed to specify the initial compartment occupancies. We have again specified a population of 1,000 individuals, but in this case we have split them evenly between the two locations and chosen 100 individuals in location 0 to be infected at the start of the simulation period.

Figure 3 shows the simulated dynamics of the infected populations at each of the two locations for each of the five trajectories generated by MASTER and deposited in its output file. The structure of the population leads to a clear delay in the progress of the epidemic in location 1 relative to that in location 0, precisely as one would expect.

Stochastic Models of Viral Infection

Up to this point, we have given no consideration at all to how the simulated trajectories produced by MASTER are generated. The default behavior of MASTER is to use Gillespie's direct method (Gillespie 1976), also known as the Stochastic Simulation Algorithm or SSA. This is often regarded as an "exact" method as it is capable of drawing trajectories from the true solution to the master equation presented earlier as equation (17). However, the computational complexity of this method has a polynomial dependence on the size of the population. This is due to the fact that it explicitly simulates every reaction event and the rate of occurrence those events is a function of the population size through equation (18). Thus, when models involve large numbers of interacting individuals, another approach is necessary.

To address this issue, MASTER allows the user to specify that the simulation should be conducted using either the τ-leaping method developed by Gillespie (2001), or the step anticipation τ-leaping (SAL) algorithm of Sehl et al. (2009).

We will demonstrate the use of this feature by considering the basic model of within-host viral infection dynamics (Perelson et al. 1993; Nowak and May 2000; Perelson 2002).

This model involves three distinct "populations": uninfected cells denoted X, productively infected cells Y and free-floating viral particles (virions) V. Using the reaction formalism, the stochastic form of this model can be written (Vaughan et al. 2011):

$$0 \xrightarrow{\lambda} X \tag{7}$$

$$X + V \xrightarrow{\beta} Y \tag{8}$$

$$Y \xrightarrow{k} Y + V \tag{9}$$

$$X \xrightarrow{d} 0 \tag{10}$$

$$Y \xrightarrow{a} 0 \tag{11}$$

$$V \xrightarrow{u} 0 \tag{12}$$

The MASTER specification of this model can be found in Appendix (see Stochastic Within-Host Viral Infection Model). It follows the same general form as that of the unstructured SIR model. Use of the τ-leaping algorithm is specified by the addition of the following inside the `<run>` element:

```
<stepper spec='TauLeapingStepper'
         timeStep='0.01'/>
```

The time step is in units of simulation time, and should be selected carefully. A sensible procedure is to reduce the time step size until reducing it further produces no discernible change in the results. To derive a quantitative understanding of the stochastic variation in the results, we can use the "Ensemble Summary" simulation type. Just as for the "Ensemble" type used in the previous section, this requires that the value of the `nTraj` attribute of the `<run>` element be set to the number of individual trajectories to be generated. Additionally, however, the "Ensemble Summary" simulation type requires the user to define moments to be estimated from those trajectories. Suppose that we are interested in the dynamics of the expected sizes of the infected cell and virus populations, and the dynamics of the covariance between those two populations. We might then include the following elements inside the `<run>` element:

```
<moment spec='Moment' momentName='Y'>
  <factor idref='Y'/>
</moment>
<moment spec='Moment' momentName='V'>
  <factor idref='V'/>
</moment>
<moment spec='Moment' momentName='YV'>
  <factor idref='Y'/>
  <factor idref='V'/>
</moment>
```

The `<factor>` elements specify individual population sizes to multiply together to form the contribution to the moment estimate from each trajectory, as described in later section. The third `<moment>` element thus specifies that a moment with the name "YV" formed as the product of the sizes of the unstructured populations Y and V should be estimated. Together, these elements thus instruct MASTER to produce estimates of $E(N_Y(t))$, $E(N_V(t))$, and $E(N_Y(t)N_V(t))$.

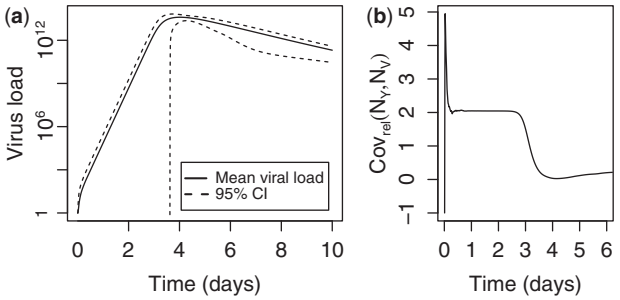

Fig. 4. Using MASTER to perform within-host infection dynamics simulations. (*a*) Expected viral load conditional on chronic infection. (*b*) Relative covariance between infected cell and virion within-host populations.

Finally, it may be desirable to exclude simulated trajectories in which chronic infection fails to occur. The effect of such exclusion can be very strong when starting from small numbers of infected cells or virions, which can easily be driven to extinction by demographic noise. To condition the results on cases where the infection takes hold, we can place the following population size end condition element inside the `<run>` element:

```
<populationEndCondition spec='PopulationEndCondition'
                        threshold="0"
                        exceedCondition="false"
                        isRejection="true">
  <population idref='Y'/>
  <population idref='V'/>
</populationEndCondition>
```

This end condition is met when the *sum* of the sizes of the populations specified by the `<population>` elements becomes equal to zero. The `isRejection` attribute causes trajectories that meet the condition at any point during the simulation to be immediately discarded.

Figure 4 illustrates the moment estimates obtained by MASTER using 10^3 trajectories, with reaction rates chosen from a previous publication (Vaughan et al. 2011) and an initial virus count of 1. Figure 4*a* shows that the absolute number of virions present in the trajectories at the time of peak viremia is on the order of 10^{13}, strongly justifying our use of τ-leaping. The effect of the conditioning on the survival of the infection is evident in the tight confidence intervals about the population size at this peak; although there remains a wide uncertainty in the viral load before the peak. Figure 4*b* shows the dynamics of the estimated relative covariance between the infected cell and virion populations, which we define as

$$\mathrm{Cov}_{\mathrm{rel}}(N_Y(t), N_V(t)) = \frac{E(N_Y(t)N_V(t))}{E(N_Y(t))E(N_V(t))} - 1 \tag{13}$$

and can calculate based on MASTER's estimates of $E(N_Y(t)N_V(t))$ together with those of the individual means. The strong pre-peak-viremia correlation between the demographic fluctuations in each of these populations noted elsewhere (Vaughan et al. 2011) is clearly visible.

Genealogy Simulation

We finally turn to the use of MASTER as a stochastic simulator of phylogenetic trees and networks. The software is capable of introducing special individual population members at arbitrary times during a simulation and keeping track of the parent–child relationships of direct descendants of those individuals. This is a subtly different problem to that of using stochastic simulation to solve master equations such as equation (17), as it requires specification of additional information beyond that contained in reactions such as equation (19). In particular, although such reactions are sufficient to discern the effect an event has on the sizes of the populations involved, it says nothing about the actual inheritance relationships between the individual reactants and products.

Automatic Parentage Allocation

MASTER deals with this issue in two ways. First, it contains an algorithm for automatically ascribing inheritance relationships between pairs of reactant and product individuals. This is as follows:

1) Select the first reactant individual on the left-hand side of the reaction.
2) Assign all product individuals which do not already have parents and have the same population *type* as the selected reactant to be children of that reactant.
3) Select the reactant to the right of the previous one and go back to step 2, repeating until either no reactants remain or no orphan products remain.

This "greedy" algorithm always produces tree-like relationships (every child has only one parent), which is often what is desired for forward-time birth–death models.

As an example of the usefulness of this approach, consider a birth–death model involving a single population of individuals X subject to the following pair of reactions:

$$X \xrightarrow{b} 2X \tag{14}$$

$$2X \xrightarrow{d} X \tag{15}$$

This is one example of a stochastic logistic model, and as such describes a population with a carrying capacity. Note that this model only contains competitive death and not spontaneous decay, with the result that complete extinction of the population is impossible.

The XML script that we use to draw the tree of descendants from an individual chosen from this population is listed in Appendix (see Stochastic Logistic Tree Simulation). The form of the model specification is very similar to those we have already seen; the only major difference is that the `spec` attributes have been changed to reflect the fact that we are now dealing with a model and reactions, which specify

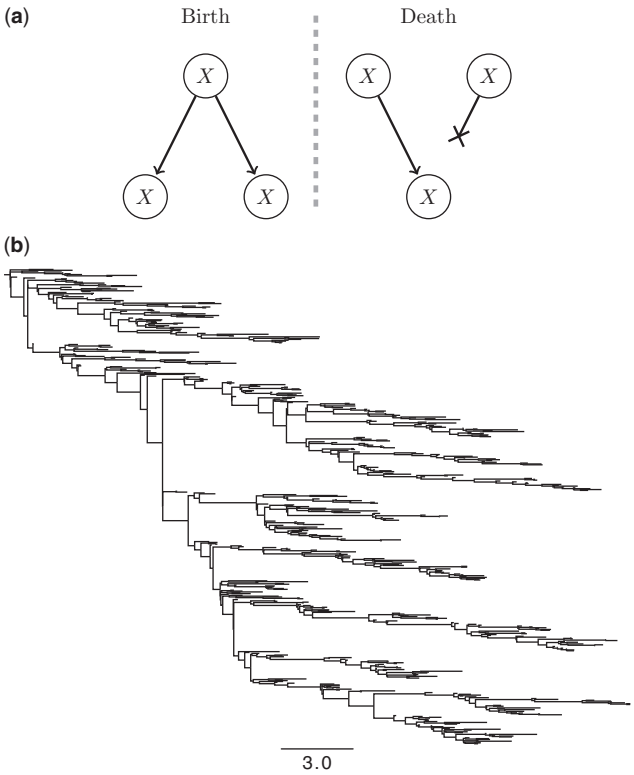

Fig. 5. A stochastic logistic model with inheritance tracking. (*a*) Inheritance relationships between reactants (top) and products (bottom). (*b*) A typical tree produced by MASTER.

inheritance relationships. We have chosen reaction rates which correspond to a population with a carrying capacity of 100.

Importantly, the reactions in this model specification use precisely the same nomenclature as those in MASTER input files for simulations which do not track inheritance relationships. This is the signal for MASTER to assign its own parent–child relationships according to its algorithm. In this case, the algorithm results in the relationships shown in figure 5*a*: the birth reaction assigns both products as children of the single reactant, whereas the death reaction assigns the sole reactant as the child of the first parent, leaving the lineage of the second reactant to terminate.

The specification of the initial state of the simulated system is perhaps what is most visibly different between this input file and those we have seen previously:

```
<initialState spec='InitState'>
  <populationSize spec='PopulationSize'
                  population='@X' size='9'/>
  <lineageSeed spec='Individual'
               population='@X'/>
</initialState>
```

Here, the `<populationSize>` element is used to specify the initial number of individuals in the X population whose descendant trees will *not* be recorded. If this were the only component of the `<initialState>` element the simulation would proceed without any inheritance tracking at all. The additional `<lineageSeed>` element specifies

an *additional* individual to include in the initial population whose descendants will be traced.

Finally, we need to specify what to do with the simulated tree once constructed. As discussed previously we have the option of producing either Newick or NEXUS output. As NEXUS allows for additional annotation, which MASTER uses to store details of which reaction is associated with each node, we opt for that here by including the following element within the `<run>` element:

```
<output spec='NexusOutput'
        fileName='BirthDeathTree.nexus'/>
```

A typical tree produced by this simulation is shown in figure 5*b* (the graphic was produced using FigTree Rambaut [2012]). The specification we have provided is fairly likely to produce trees as large or larger than this, due to the initial population bottleneck: there is a 10% chance that the descendants of the seed individual will fix in the population.

Manual Parentage Allocation

Alternatively, MASTER allows users to manually specify inheritance relationships between reactant and product individuals. We can demonstrate the utility of this by using MASTER to simulate a tree conditional on a given number of taxa under the coalescent process (Kingman 1982). This process is a simple Markovian decay process in the backward-time direction for the number of lineages L existing at a given time:

$$2L \xrightarrow{\chi} L. \tag{16}$$

Here $\chi = (2N_e g)^{-1}$, where N_e is the effective size of the haploid population in which the sampled lineages are embedded, and g is its generation time. In terms of building a tree, the coalescent process proceeds in reverse time: starting with n leaves, successive coalescent events (of which there are $n - 1$) join lineages together until finally only one remains. Thus, reactant pairs need to jointly "parent" a single "child" product. This is not a tree-like structure when read in the direction of the process, implying that the automatic allocation discussed previously will not produce the required relationships.

To direct MASTER to simulate this process with the required relationships, we use the following reaction specification for a coalescent process under a scaled effective population size $N_e g = 0.5$:

```
<reaction spec='InheritanceReaction'
          reactionName='Coalescence'
          rate='1.0'>
  L:1 + L:1 → L:1
</reaction>
```

The ":1" following each reactant population identifier causes MASTER to regard those reactants as parents of each of the reaction products which carry the same suffix. We could have used any integer following the colon, as long as the same integer is appended to all products and reactants which are to be regarded as parents/children of one another. This particular reaction specification results in the inheritance relationships shown in figure 6*a*.

Initialization of this simulation amounts to adding multiple individuals whose "descendants" be tracked to form the leaves of the coalescent tree. In this case, we use the following state initializer:

```
<initialState      spec='InitState'>
  <lineageSeedMultiple
                   spec='MultipleIndividuals'
                   population='@L' copies="20"/>
  <lineageSeed spec='Individual'
                   population='@L' time="1"/>
  <lineageSeed spec='Individual'
                   population='@L' time="2"/>
  <lineageSeed spec='Individual'
                   population='@L' time="3"/>
</initialState>
```

The first line uses a short-hand notation for specifying 20 lineage seeds at the start of the simulation. The following three lines demonstrate how lineages can be seeded at times other than the start of the simulation. In the case of the coalescent, these lineages can represent serially or noncontemporaneously sampled taxa (Rodrigo and Felsenstein 1999). Note that in this calculation, we track the inheritance of *all* individuals present in the simulation, so there is no need for additional `<populationSize>` elements. (This may seem at odds with the assumptions of the coalescent, which specifies statistical properties of ancestral trees of individuals sampled from a much larger population. However, the dynamics of the larger population are not modeled *stochastically* by the coalescent, which absorbs the influence of that population into the rate parameter χ.)

Finally, in the case of reverse-time coalescent tree simulations, it makes particularly good sense to use a lineage-dependent end condition to terminate the calculation rather than specifying a fixed time. To accomplish this in MASTER, we include the following inside the `<run>` element, which terminates the calculation when a single lineage remains (i.e., when the most recent common ancestor of the sampled leaf individuals is reached):

```
<lineageEndCondition spec='LineageEndCondition'
                     nLineages="1"/>
```

The full input file for this example is listed in Appendix (see Serially Sampled Coalescent Tree Simulation). Figure 6*b* illustrates a typical generated tree. This figure was produced using FigTree to visualize a NEXUS file generated using an `<output>` element having `spec` value `NexusOutput` and the `reverseTime` attribute set to "true". (Failure to set this attribute causes the resulting NEXUS file to contain an extended Newick representation of an upside-down tree.) It shows the standard coalescent structure on the right, with the modifications to this structure wrought by the "serially sampled" leaves at times 1, 2, and 3 before the time of the contemporaneous leaves. These additional leaves tend to increase the age of the tree root.

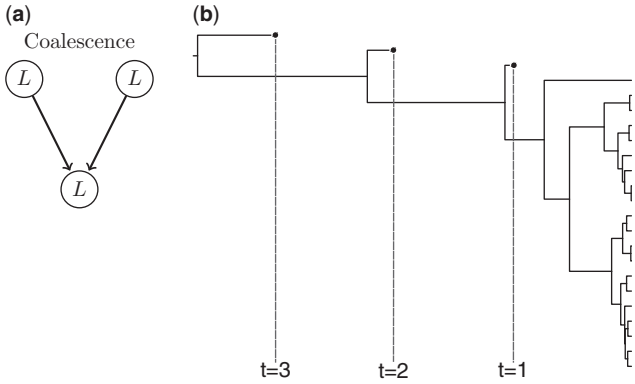

FIG. 6. Simulating a serially sampled coalescent tree. (*a*) Inheritance relationships between coalescing individuals. (*b*) A typical tree generated using the chosen leaf times and coalescence rate. Note the late introduction of lineages at times $t = 1, 2, 3$, extending the age of the root substantially.

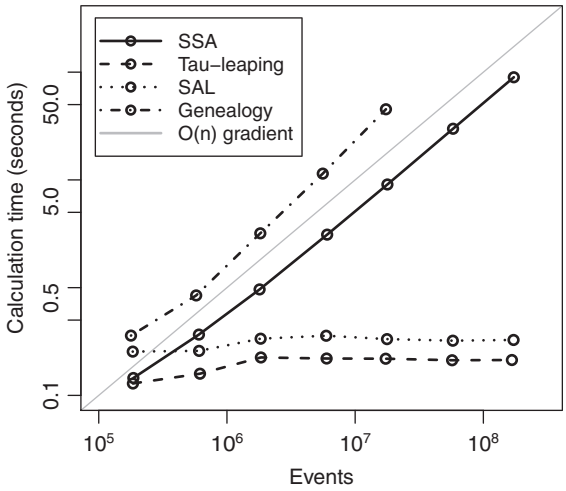

FIG. 7. Elapsed computation times used in simulating population dynamics under a stochastic logistic model for 100 units for a range of total event counts (selected by adjusting the carrying capacity), using each of the calculation methods currently implemented in MASTER. The gray line depicts the gradient associated with strict linear dependence on the number of events simulated. Tau-leaping and SAL algorithms were simulated using a fixed time step $\tau = 0.001$ (computations performed on an Intel Core i5 CPU operating at 3.0 GHz).

Computational Efficiency of MASTER

We note that an important factor in determining the usefulness of a given piece of simulation software is the computational efficiency of the program. Here, we use the term efficiency to mean the degree to which the program uses either CPU cycles (time) or available digital memory (space) effectively. The efficiency depends primarily on the computational complexity of the algorithm used to perform the simulation. In practice, however, it can also depend on the specifics of the implementation such as the combination of programming language and compiler used to express it.

Figure 7 displays a comparison between run times for each of the algorithms MASTER implements when applied to the stochastic logistic model described in earlier section. The population size is initialized to 1 and the model propagated for 100 time units, with the total number of simulated events varied by selecting different carrying capacities. We clearly see that the time complexity under Gillespie's SSA is linear with respect to the total event count, in stark contrast to the fixed time-step implementations of the approximate τ-leaping and SAL algorithms which have no explicit dependence on the number of reactions that fire. Inheritance tracking is performed using an extension of the standard SSA and thus displays nearly equivalent $O(n)$ scaling to that algorithm, although there is an additional time penalty associated with updating the tree.

The reader should be aware that the running times shown in figure 7 for the approximate algorithms were generated for fixed time-steps of 10^{-3} time units. Thus, the comparison is not entirely "fair" as it fails to take into account the fact that the SSA results are exact draws from the true trajectory probability distribution, whereas the time step size in the approximate algorithms required to achieve a given absolute accuracy may in general be an increasing function of the population size. This is reflected in the form of the step size selection function for the full adaptive τ-leaping algorithm (Gillespie 2001).

In contrast to running time, the memory usage (i.e., space complexity) of MASTER depends not only the algorithm used but also which of the specific calculation types and output mechanisms is used. For example, the "Ensemble" calculation type uses vastly more memory when generating a given number of trajectories than the "Ensemble Summary" type, as the former records all calculated trajectories in memory before writing them to disk while the latter only records the accumulating moment estimates (as noted previously, this is precisely the reason why we include this calculation type).

Users should be aware that lineage tracking in MASTER can be *very* memory intensive, as each reaction event involving tracked lineages results in a new node in the graph (the largest genealogy simulations used to generate figure 7, for instance, resulted in inheritance trees with nearly 2×10^7 nodes). This is something the user should keep in mind if out-of-memory errors are encountered, and is one of the reasons that the capability of tracking the genealogy of a population subset has been included.

Discussion

MASTER is a new tool that we believe substantially improves the ease and accuracy with which reasonably complex stochastic simulations of population dynamics can be conducted. In the earlier examples, we have sought to give a feel for the kinds of simulations MASTER can handle, including structured and unstructured epidemiological models, within-host virus infection models. We have also shown how MASTER bridges the gap between software, which efficiently simulates the stochastic dynamics of large populations

and software which is capable of generating phylogenetic trees under the same models.

MASTER has additional capabilities beyond those exemplified in this article. These include the ability to simulate phylogenetic networks and other non-tree-like inheritance structures. A limited amount of post-processing on simulated genealogies can also be performed, such as pruning all nodes besides those ancestral to a random subset of leaves, simulating the effect of sampling bias on the shape of the simulated genealogy. Details regarding these and other features can be found on the project's web page.

Materials and Methods

Mathematical Background

Here, we describe the actual mathematical problem that MASTER sets out to solve, as well as introducing nomenclature MASTER uses when dealing with structured populations.

Master Equations and Reaction Schemes

Fundamentally, MASTER is designed to simulate models composed of a fixed number of populations. The true state of the system at a particular time t is denoted $\vec{N}(t)$, where the vector is composed of integer elements representing the sizes of the constituent populations. As we consider stochastic models, the precise value of $\vec{N}(t)$ is unknown. Instead, we deal with the probability distribution over possible values of the state vector, that is, $P(\vec{N}(t) = \vec{n})$.

The dynamical models we consider are continuous-time Markovian stochastic models described by the following master equation:

$$\frac{\partial}{\partial t} P(\vec{n},t) = \sum_k \left[T_k(\vec{n} - \vec{\nu}_k) P(\vec{n} - \vec{\nu}_k,t) - T_k(\vec{n}) P(\vec{n},t) \right], \tag{17}$$

where we have used the short-hand notation $P(\vec{n},t) \equiv P(\vec{N}(t) = \vec{n})$. The index k represents individual stochastic reactions, each of which are specified by a vector ν_k and a propensity function $T_k(\vec{n})$. The vector $\vec{\nu}_k$ specifies the change to the system state that occurs when reaction k "fires". For instance, if reaction k is the only reaction which fires in the time interval $[t, t+\delta]$, we can say that $\vec{N}(t+\delta) = \vec{N}(t) + \vec{\nu}_k$. The propensity function $T_k(\vec{n})$ describes the average rate at which reaction k will fire given a particular system configuration $\vec{n}$. It takes the form

$$T_k(\vec{n}) = \lambda_k \prod_i n_i^{\underline{\mu_k^i}} \tag{18}$$

where λ_k is the state-independent reaction rate constant, i labels individual population elements of $\vec{n}$ and μ_k^i is the number of individuals from population i involved in this reaction. The under-bar represents the falling factorial: $n_i^{\underline{\mu_k^i}} = n_i(n_i - 1)\ldots(n_i - \mu_k^i + 1)$.

Chemistry provides us with a particularly useful notation for describing stochastic models of this kind. We can express the components of reaction k relevant to our stochastic model in the following way:

$$\sum_i \mu_k^i X_i \xrightarrow{\lambda_k} \sum_j \nu_k^j X_j. \tag{19}$$

The symbol X_i represents an individual chosen from population i. The left-hand side specifies the numbers μ_k^i of each type i of individual, which must be in contact in order for the reaction to proceed. These individuals are referred to as the reactants. The right-hand side specifies the numbers ν_k^j of each type j which are produced by the reaction. These individuals are referred to as the reaction products. Knowledge of μ_k^i, ν_k^j, and λ_k is sufficient to reconstruct the master equation, as $T_k(\vec{n})$ depends only on λ_k and μ_k^i and the elements of the state change vector $\vec{\nu}_k$ can be obtained directly from $\nu_k^j - \mu_k^i$.

MASTER allows one to specify a stochastic model in terms of a collection of individual reactions of this form, using a simple syntax very similar to the chemical reaction form given earlier.

Population Structure

Many populations in the real world are structured in some way. This structuring can be due to geographical constraints, but it can also signify other logical partitions of the full population.

As far as the mathematics is concerned, there is little fundamental difference between a collection of distinct yet connected populations and a single "structured" population containing the same number of subpopulations. However, in specifying a simulation and interpreting the output, it is very useful to be able to refer to individual populations within a larger meta-population in a way that is at least reminiscent of the real-world relationship between the components.

MASTER achieves this goal by grouping all populations within the scope of the model into one or more population types. The populations within each type are arranged into a $|\vec{d}|$-dimensional array where $\vec{d}$ is a vector of positive integers specifying the size of the array in each dimension. The general effect of this is that both a type name and a cell location within the corresponding array are necessary to specify a particular population in a MASTER model. The relationship between populations and population types is illustrated in figure 8.

Note that although MASTER internally represents single isolated populations in terms of a zero-dimensional population type, its XML interface provides a short-hand notation for this. In other words, users need not usually deal directly with population types if their model does not involve population structure.

XML Syntax Overview

MASTER reads the specification of the model from a XML input file and uses this to determine what calculations to run and what output to write. In this section, we will present a general overview of the syntax of these files.

The general outline of the structure of a MASTER input file is shown in figure 9. As noted in the introduction, MASTER operates as an add-on to BEAST 2 (BEAST 2 development team 2013). It thus uses an extension of the XML format of that software. As a result, every MASTER input file opens with a `<beast>` element specifying the version of the input file format and location of the Java classes that the file will refer to. This first line and the closing tag `</beast>` can be copied verbatim into all standard MASTER simulation specification files. Every element within the XML file must define the attribute `spec`. This attribute specifies the actual MASTER entity (BEAST plug-in) associated with that element. Additionally, any element may be given the attribute `id='idString'`, which causes BEAST to use "idString" to uniquely identify this element in the XML file. The contents of this element can then be referred to by other elements through the `idref` attribute. Additionally, the value of attributes can be set to point to the identified element by using `attribute='@idString'` syntax. Further information about the general form of BEAST 2 XML can be found on the BEAST 2 project web page.

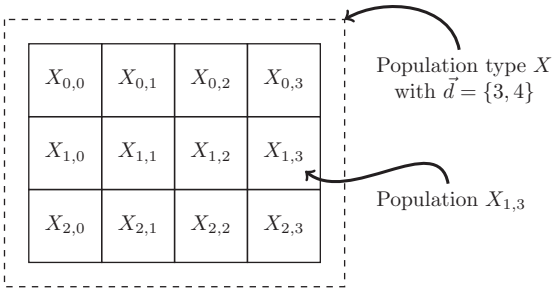

Fig. 8. Relationship between populations and population types in MASTER.

The `<run>` element specifies the actual simulation. In this case, the value of `spec` attribute specifies the particular kind of simulation which will be run. Possible values are "Trajectory," "Ensemble," "EnsembleSummary," "InheritanceTrajectory," and "InheritanceEnsemble"; corresponding to the simulation types discussed in the introduction. Each of these simulation types is associated with a set of options, which are specified as additional attributes of the `<run>` element and are described in detail on the website. There are three attributes that are valid for all simulation types, however. The first is the `simulationTime` attribute, which takes a numeric value and is used to specify the time at which the simulation will end. It is mandatory in the case of the "Ensemble Summary" simulation type, as moment estimation requires many trajectories of equal length. In all other cases it is optional, as termination may be brought about via state-dependent end conditions to be discussed later.

The second common attribute is `nSamples` which, if specified, determines the number of evenly spaced times at which population sizes (or moment estimates) should be recorded. As with `simulationTime`, this is mandatory for "Ensemble Summary" calculations but optional for all other simulation types. The default behavior when it is omitted from those types varies, but generally causes the simulator to record population sizes at times determined by the simulation algorithm in use (i.e., at each reaction time if the default stochastic simulation is used, or following each integration interval if either of the finite time-step algorithms described later are used).

The final common attribute is `verbosity`, which takes integer values in the range from 0 to 2. This attribute controls the level of detail in the messages displayed by MASTER indicating the progress of the calculation, with 0 entirely preventing the display of such messages.

```
<beast version='2.0' namespace='beast.core.parameter:master.beast'>

 <run spec='Trajectory
            |Ensemble
            |EnsembleSummary
            |InheritanceTrajectory
            |InheritanceEnsemble'
      ... [options] ... >

  <model spec='Model
             |InheritanceModel'>
    <!-- Model specification -->
  </model>

  <initialState spec='InitState'>
    <!-- Initial system state specification -->
  </initialState>

  <!-- End conditions -->

  <!-- Outputs -->

 </run>
</beast>
```

Fig. 9. General structure of a MASTER XML file.

Algorithm Selection

For simulation types that do not allow for genealogy tracking, any one of three algorithms may be used to generate the stochastic trajectories. If none is specified, the default behavior is to use the direct method (Gillespie 1976). Otherwise, including the `<stepper>` element with `spec` value `TauLeapingStepper` inside `<run>` can be used to select τ-leaping as the preferred algorithm. This implementation uses a fixed time-step set using the attribute `stepSize`. Likewise, the step anticipation τ-leaping (SAL) algorithm of Sehl et al. (2009) can be selected by replacing the `spec` value with `SALStepper`.

Model Specification

A stochastic population dynamical model in MASTER is composed of

1) a list of named population types, each associated with one or more populations, and
2) a list of reactions or reaction groups involving individual members of those populations.

In the XML file, a model is contained within the `<model>` element which uses the `spec` value `Model`. Within this, population types are specified using the `<populationType>` element. Each population type has a name, which is specified as a string value for the `typeName` attribute, and a `dim` attribute, which takes a space-delimited list of integers specifying the dimension of the array of populations contained within this type.

In the instance that a single isolated population with no internal structure is required, a `<population>` element can be used in place of `<populationType>`. A name is still required, although in this case it is specified using the string value of the `populationName` attribute. The effect of using this element is still to add a population type to the model having the specified name, but one which has only a single constituent population.

The reactions that give rise to dynamics in the model are specified individually using the very readable syntax of the `<reaction>` element, which has the `spec` value `Reaction`. For instance, an exponential birth process, which proceeds by taking an individual belonging to population {1,2} of type X and duplicating it according to the reaction

$$X_{1,2} \xrightarrow{\lambda} 2X_{1,2} \tag{20}$$

with $\lambda = 1$ can be included in the model using the element:

```
<reaction spec='Reaction'
          reactionName='birth'
          rate='1.0'>
  X[1,2] → 2X[1,2]
</reaction>
```

Note that in the case of "Inheritance Trajectory" and "Inheritance Ensemble" simulations, a slightly different syntax must be used allowing for the specification of inheritance relationships between reactant and product individuals. This syntax is described in earlier section.

It is sometimes sensible to group similar reactions together. One example is a model involving migration of individuals between island populations. In this case, it would make sense to group together the reactions specifying migrations between all possible pairs of island populations. This can be achieved using the `<reactionGroup>` element (`spec` value `ReactionGroup`), which allows the user to name such a reaction group using the `reactionGroupName` attribute and can itself contain an arbitrary number of individual `<reaction>` elements.

Initial States

To begin a dynamical simulation, the initial state of the system must be specified. This is achieved through the use of the `<initialState>` element, which resides inside the `<model>` element. For simulations dealing only with population size dynamics, the initial state is completely defined by the starting sizes of each of the populations in the model. To specify these starting population sizes, `<populationSize>` elements (with `spec` value `PopulationSize`) are added to `<initialState>`. Each of these elements has a `size` attribute which specifies an initial population size, as well as a `population` attribute used to tie the size to a particular population. The population itself is in turn specified with its own element, `<population>` (`spec` value `Population`) which uses the attributes `type` and `location` to specify a particular MASTER population (further details are available in the examples which follow).

In the case of "Inheritance Trajectory" simulations, additional special individuals are added to the starting populations using the `<lineageSeed>` element with `spec` attribute value `Individual`. Just as for `<populationSize>`, these elements have an attribute `population` which specifies the population that the individual is to be added to. By default, these individuals are added at the very start of the simulation. However, the attribute `time` can be used to specify their addition at particular times following the start of the simulation. This is particularly useful for the simulation of coalescent trees from serially sampled data.

Estimating Moments

As noted previously, the goal of "Ensemble Summary" simulations is to use a large number of trajectories to determine the dynamics of estimated moments of the population size variables. As the calculation of the moments/summary statistics is done during the course of the calculation, this is much more memory efficient than combining the trajectories after they have been generated using the "Ensemble" simulation style.

An "Ensemble Summary" simulation specification is essentially the same as one specifying an "Ensemble" simulation, with the addition of one or more `<moment>` or `<momentGroup>` elements. The `<moment>` element (with `spec` value `Moment`) has a `momentName` attribute which is used to specify a unique name for the moment.

Each <moment> element must contain one or more <factor> elements which are linked to individual populations either via the idref attribute or through directly specifying a population with the Population spec value. For each time point at which the population sizes are sampled, populations specified by <factor> elements are multiplied together and the product averaged across all trajectories to obtain the moment estimate.

If more general polynomial moments of the population sizes are required, the <momentGroup> element (with spec value Moment) can be used. This takes a unique name via the momentGroupName attribute and can itself include one or more <moment> elements. An additional Boolean attribute sum can be set to true to cause the individual estimates from each of the <moment>s contained int the group to be summed together. In the case that sum is missing or set to "false", the <momentGroup> element merely provides a means of logically grouping similar moments together in a way that is reflected in structure of the output.

State-Dependent End Conditions

MASTER allows for one or more state-dependent end conditions to be set. These end conditions result in immediate cessation of the simulation of a trajectory, regardless of the time at which the condition is met. Then, depending on the type of end condition, the trajectory can be accepted as a completed calculation or it can be rejected and the calculation begun anew.

End conditions come in two flavors. First, end conditions which depend only on the total number of individuals present in a particular population are specified using one or more <populationEndCondition> elements (each having spec value PopulationEndCondition). Within each of these elements, one or more <population> elements must be provided. These indicate those populations whose sizes are summed together to produce the conditioning value. The attribute threshold specifies the magnitude of the population size sum at which the condition comes into effect. Additionally, a Boolean attribute exceedCondition is used to specify whether the condition is met by exceeding or dipping below that threshold value.

A final attribute to this element is the Boolean isRejection attribute, which specifies whether trajectories meeting this criterion are to be discarded or are accepted as a result of the simulation. Note that all end conditions must result in rejection in the case of "Ensemble Summary" simulations, as these calculations require generation of many trajectories of exactly the same length.

The second flavor of end condition is specific to the "Inheritance Trajectory" simulation type, and depends on the number of lineages remaining. These are specified using instances of the <lineageEndCondition> element (spec value LineageEndCondition). This element takes the attribute nLineages, which is used to set the number of remaining lineages at which the condition is met, and the attribute isRejection, which behaves in the same way as for population size end conditions. The optional attribute population can be used to specify that the condition applies only to lineages belonging to a particular population.

Output Files

The final component of the MASTER XML file is the specification of the form the output should take. In all cases besides "Inheritance Trajectory" simulations there is currently a single option: to have the results written to a file using the hierarchical JSON format. Although not traditionally used as a scientific data format, JSON is both versatile and portable, with parsers available for many commonly used numerical analysis and visualization platforms including R, Matlab, and Python. This output is specified using the jsonOutput element (spec value JsonOutput) whose single attribute fileName sets the name of the file the output will be written to.

For "Inheritance Trajectory" calculations, there are two possible output options. The first is to have the simulated inheritance graph written to a file in extended Newick format (Cardona et al. 2008). This is similar to the standard Newick format for representing phylogenetic trees, but applies equally well to phylogenetic networks. Note that in the case that the simulated network is tree-like, the extended Newick representation reduces to the standard Newick format. This output option is specified using the <newickOutput> element (spec value NewickOutput) which, in addition to the fileName attribute, takes a Boolean reverseTime attribute. This attribute causes the output generator to traverse the simulated network in the backward time direction, allowing networks which are tree-like in this direction to be expressed using the standard Newick format rather than the extended format.

The second output specifically applicable to "Inheritance Trajectory" calculations is the Nexus format. This format of output file contains an (extended) Newick representation of the graph with detailed annotations specifying the names of the reactions corresponding to each of the nodes. In the case of a tree-like network, this file can be read by the software FigTree and used to generate detailed visualizations of the inheritance process. This output option is specified using the <nexusOutput> element (spec value NexusOutput) which takes exactly the same attributes as the <newickOutput> element.

Note that the <jsonOutput> element is still valid in the case of an "Inheritance Trajectory" calculation, but the population sizes will only be recorded if this behavior is specifically requested in the simulation-specific attributes of <run>.

Acknowledgments

This work was supported by a postdoctoral fellowship from the Allan Wilson Centre for Molecular Ecology and Evolution to T.G.V. and by a Rutherford Discovery Fellowship from the Royal Society of New Zealand to A.J.D.

Appendix

MASTER Input Files

This appendix contains listings of the MASTER XML input files used in all examples bar the first, which is listed in figure 1.

Structured Epidemiological Model

```
<beast version='2.0' namespace='master.beast:beast.core.parameter'>
  <run spec='Ensemble'
       simulationTime='50'
       nTraj="5">
    <model spec='Model' id='model'>
      <populationType spec='PopulationType' id='S' typeName='S' dim="2"/>
      <populationType spec='PopulationType' id='I' typeName='I' dim="2"/>
      <population spec='Population' id='R' populationName='R'/>

      <reactionGroup spec='ReactionGroup' reactionGroupName="Infection">
        <reaction spec='Reaction' rate="0.001">
          S[0] + I[0] → 2I[0]
        </reaction>
        <reaction spec='Reaction' rate="0.001">
          S[1] + I[1] → 2I[1]
        </reaction>
      </reactionGroup>
      <reactionGroup spec='ReactionGroup' reactionGroupName="Recovery">
        <reaction spec='Reaction' rate="0.2">
          I[0] → R
        </reaction>
        <reaction spec='Reaction' rate="0.2">
          I[1] → R
        </reaction>
      </reactionGroup>
      <reactionGroup spec='ReactionGroup' reactionGroupName="Migration">
        <reaction spec='Reaction' rate="0.01">
          S[0] → S[1]
        </reaction>
        <reaction spec='Reaction' rate="0.01">
          S[1] → S[0]
        </reaction>
        <reaction spec='Reaction' rate="0.01">
          I[0] → I[1]
        </reaction>
        <reaction spec='Reaction' rate="0.01">
          I[1] → I[0]
        </reaction>
      </reactionGroup>
    </model>

    <initialState spec='InitState'>
      <populationSize spec='PopulationSize' size='400'>
        <population spec='Population' type='@S' location="0"/>
      </populationSize>
      <populationSize spec='PopulationSize' size='500'>
        <population spec='Population' type='@S' location="1"/>
      </populationSize>
      <populationSize spec='PopulationSize' size='100'>
        <population spec='Population' type='@I' location="0"/>
      </populationSize>
    </initialState>
    <output spec='JsonOutput' fileName='StructuredSIR_output.json'/>
  </run>
</beast>
```

Stochastic Within-Host Viral Infection Model

```
<beast version='2.0' namespace='master.beast:beast.core.parameter'>
  <run spec='EnsembleSummary'
       nSamples="1001"
       nTraj="1000"
       simulationTime='10'>

    <stepper spec='TauLeapingStepper' stepSize='0.01'/>
    <model spec='Model' id='model'>
      <population spec='Population' id='X' populationName='X'/>
      <population spec='Population' id='Y' populationName='Y'/>
      <population spec='Population' id='V' populationName='V'/>

      <reaction spec='Reaction' reactionName="CellBirth" rate="2.5e8">
        0 → X
      </reaction>
      <reaction spec='Reaction' reactionName="Infection" rate="5e-13">
        X + V → Y
      </reaction>
      <reaction spec='Reaction' reactionName="NewVirion" rate="1e3">
        Y → Y + V
      </reaction>
      <reactionGroup spec='ReactionGroup' reactionGroupName="Death">
        <reaction spec='Reaction' rate="1e-3">
          X → 0
        </reaction>
        <reaction spec='Reaction' rate="1">
          Y → 0
        </reaction>
        <reaction spec='Reaction' rate="3">
          V → 0
        </reaction>
      </reactionGroup>
    </model>

    <populationEndCondition spec='PopulationEndCondition'
              threshold="0"
              exceedCondition="false"
              isRejection="true">
      <population idref='Y'/>
      <population idref='V'/>
    </populationEndCondition>

    <initialState spec='InitState'>
      <populationSize spec='PopulationSize' population='@X' size='2.5e11'/>
      <populationSize spec='PopulationSize' population='@V' size='1'/>
    </initialState>

    <moment spec='Moment' momentName='X'>
      <factor idref='X'/>
    </moment>

    <moment spec='Moment' momentName='Y'>
      <factor idref='Y'/>
    </moment>

    <moment spec='Moment' momentName='V'>
      <factor idref='V'/>
    </moment>

    <moment spec='Moment' momentName='YV'>
      <factor idref='Y'/>
      <factor idref='V'/>
    </moment>

    <output spec='JsonOutput' fileName='HIV_output_cond.json'/>
  </run>
</beast>
```

Stochastic Logistic Tree Simulation

```
<beast version='2.0' namespace='master.beast:beast.core.parameter'>
  <run spec='InheritanceTrajectory'
       simulationTime="100"
       samplePopulationSizes="true"
       verbosity='1'>
```

```
    <model spec='InheritanceModel'>
      <population spec='Population' populationName='X' id='X'/>
      <reaction spec='InheritanceReaction' reactionName='Birth' rate="1.0">
        X → 2X
      </reaction>
      <reaction spec='InheritanceReaction' reactionName='Death' rate="0.01">
        2X → X
      </reaction>
    </model>

    <initialState spec='InitState'>
      <populationSize spec='PopulationSize' population='@X' size='9'/>
      <lineageSeed spec='Individual' population='@X'/>
    </initialState>

    <output spec='NexusOutput' fileName='out.nexus'/>
    <output spec='JsonOutput' fileName='out.json'/>
  </run>
</beast>
```

Serially Sampled Coalescent Tree Simulation

```
<beast version='2.0' namespace='master.beast:beast.core.parameter'>
  <run spec='InheritanceTrajectory'
       verbosity='2'>

    <model spec='InheritanceModel'>
      <population spec='Population' populationName='L' id='L'/>

      <!— Coalescent process with N_e*g=0.5. —>
      <reaction spec='InheritanceReaction' reactionName='Coalescence'
rate="1.0">
        2L:1 → L:1
      </reaction>
    </model>

    <initialState spec='InitState'>
      <lineageSeedMultiple spec='MultipleIndividuals' population='@L'
copies="20"/>
      <lineageSeed spec='Individual' population='@L' time="1"/>
      <lineageSeed spec='Individual' population='@L' time="2"/>
      <lineageSeed spec='Individual' population='@L' time="3"/>
    </initialState>

    <lineageEndCondition spec='LineageEndCondition' nLineages="1"/>

    <output spec='NexusOutput' fileName='out.nexus' reverseTime="true"/>
  </run>
</beast>
```

References

BEAST 2 Development Team. 2013. BEAST version 2 [Internet]. Available from: http://beast2.cs.auckland.ac.nz (last accessed March 29, 2013).

Britton T. 2010. Stochastic epidemic models: a survey. *Math Biosci.* 225: 24–35.

Cardona G, Rossell F, Valiente G. 2008. Extended Newick: it is time for a standard representation of phylogenetic networks. *BMC Bioinformatics* 9:532.

Couture-Beil A. 2013. rjson: JSON for R (version 0.2.12) [Internet]. Available from: http://cran.r-project.org/web/packages/rjson/ (last accessed March 29, 2013).

Drummond AJ, Pybus OG, Rambaut A, Forsberg R, Rodrigo AG. 2003. Measurably evolving populations. *Trends Ecol Evol.* 18:481–488.

Drummond AJ, Rambaut A. 2007. BEAST: Bayesian evolutionary analysis by sampling trees. *BMC Evol Biol.* 7:214.

Drummond AJ, Suchard MA, Xie D, Rambaut A. 2012. Bayesian phylogenetics with BEAUti and the BEAST 1.7. *Mol Biol Evol.* 29: 1969–1973.

Gillespie DT. 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J Comput Phys.* 22:403.

Gillespie DT. 2001. Approximate accelerated stochastic simulation of chemically reacting systems. *J Chem Phys.* 115:1716.

Guillaume F, Rougemont J. 2006. Nemo: an evolutionary and population genetics programming framework. *Bioinformatics* 22:2556–2557.

Hucka M, Finney A, Sauro HM, et al. (44 co-authors). 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19: 524–531.

Kermack WO, McKendrick AG. 1927. A contribution to the mathematical theory of epidemics. *Proc R Soc Lond A.* 115:700–721.

Kingman J. 1982. The coalescent. *Stochastic Process Appl.* 13:235–248.

Kühnert D, Wu CH, Drummond AJ. 2011. Phylogenetic and epidemic modeling of rapidly evolving infectious diseases. *Infect Genet Evol.* 11: 1825–1841.

Lotka A. 1920. Undamped oscillations derived from the law of mass action. *J Am Chem Soc.* 42:1595–1599.

Nowak MA, May RM. 2000. Virus dynamics. Oxford: Oxford University Press.

O'Fallon B. 2010. TreesimJ: a flexible, forward time population genetic simulator. *Bioinformatics* 26:2200–2201.

Perelson AS. 2002. Modelling viral and immune system dynamics. *Nat Rev Immunol.* 2:28.

Perelson AS, Kirschner DE, Boer RD. 1993. Dynamics of HIV infection of CD4 + T cells. *Math Biosci.* 114:81–125.

Pybus OG, Rambaut A. 2009. Evolutionary analysis of the dynamics of viral infectious disease. *Nat Rev Genet.* 10:540–550.

R Core Team. 2012. R: a language and environment for statistical computing. Vienna (Austria): R Foundation for Statistical Computing.

Rambaut A. 2012. Figtree version 1.4.0 [Internet]. Available from: http://tree.bio.ed.ac.uk/software/figtree/ (last accessed March 29, 2013).

Ramsey S, Orrell D, Bolouri H. 2005. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J Bioinformatics Comput Biol.* 3: 415–436.

Rodrigo A, Felsenstein J. 1999. Coalescent approaches to HIV population genetics. In: Crandall KA, editor. The evolution of HIV. 1st ed. Baltimore (MD): Johns Hopkins University Press.

Sanft KR, Wu S, Roh M, Fu J, Lim RK, Petzold LR. 2011. StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics* 27:2457–2458.

Sehl M, Alekseyenko AV, Lange KL. 2009. Accurate stochastic simulation via the step anticipation tau-leaping (SAL) algorithm. *J Comput Biol.* 16:1195–1208.

Swetina J, Schuster P. 1982. Self-replication with errors: a model for polynucleotide replication. *Biophys Chem.* 16:329–345.

Vaughan TG, Drummond PD, Drummond AJ. 2011. Within-host demographic fluctuations and correlations in early retroviral infection. *J Theor Biol.* 295:86.

Volterra V. 1926. Fluctuations in the abundance of a species considered mathematically. *Nature* 118:558–560.

Zanini F, Neher RA. 2012. FFPopSim: an efficient forward simulation package for the evolution of large populations. *Bioinformatics* 28: 3332–3333.