Tyler Chafin
CSCE 5013

Project final report

Purpose of project

The goal of this project was to create a forward-time DNA sequence evolution simulator, where sequences are simulated along a fixed N-taxon tree, with topology and underlying processes defined at run-time; thus program should be flexible to various simulation scenarios. Inn particular, I wanted the program to allow me to: 1) simulate divergence of lineages; 2) simulate exchange between lineages; and 3) simulate merging of lineages, at user defined times and at user defined rates where applicable.

Simulation overview

Simulation of multi-locus nucleotide data was accomplished by first random generation of “seed” states, i.e. random alleles with randomly assigned frequencies, which sum to 1 at each locus. These are then permuted for some burn-in period to create “realistic” frequency distributions and then additional lineages are created by copying and permuting the original set.

At each generation, allele frequencies are altered according to a pseudosampling method, where the frequency at time t+1 is dependant on the current frequency, the allele frequency variance, and a randomly sampled constant:

$$p^{t+1} = p + (2rnd - 1)\sqrt{3\frac{(p(1-p))}{2N_e}}$$

Where Ne is the population size. This shows that the change in frequency has an inverse relationship with population size- thus a large population will have lower changes in frequency, meaning the user can manipulate the simulation to have greater or smaller allele frequency jumps by altering the population size. Frequencies are bounded by 1 (fixation) and 0 (extinction from population).

With each iteration of the Markov chain, random mutations accumulate at a rate of μ, which is sampled for each locus from a bounded Poisson distribution of a user-defined mean. This Poisson distribution has an upper bound primarily to prevent overly long runtimes, as a single locus sampling a large mutation rate could drastically increase runtimes. These mutations assume a very simplistic Jukes-Cantor model, where all types of mutation are equally probable.

This approach introduces heterogeneity in coalescence times and gene tree topology, as would be expected under natural circumstances by sampling a distribution of simulation parameters (such as μ and thus Θ). Simulation approaches often generate datasets using fixed values, thus implicitly limiting their applicability to real data, where variance is not negligible.

Alternative demographic histories can then be imposed, and incorporated in every necessary generation, by reading in a parameter file specifying the temporal extent, and intensity, of various demographic events. For example migration can be specified as occurring between lineage 1 and 2 from generations 10,000 to 20,000 at a unidirectional rate of 0.001 individuals per generation.

Implementation

The base of the data structure of a MC simulation is the class “locus” which includes a vector of allele pools (populations) of dynamic size, each including a map (STL red-black tree, each node containing unique key) of instances of another custom class “allele_dat”, which contain allele sequences and frequencies. The STL map was chosen for the efficiency of non-terminal insertion and deletions, and relatively quick iteration, as all of these operations are performed numerous (generally millions) of times per locus simulation.

Events are read into another class “event” which contains a 2D vector table of all relevant event data, and an STL unordered_set hash table of important dates. Each generation, the class member locus::gen is checked to see if the current generation count is included in the important dates set (STL unordered_set again chosen here for fast lookup), and if so that key is queried in the event_table (two-dimensional vector), and the appropriate actions are taken depending on the type of that event (nested if-statements). For example, if the event type is "m" for migration, the given migration rate is placed into a 2 dimensional matrix of migration rates, which is queried each generation to check if migration should occur. At the end time of migration, the appropriate value in the matrix is assigned a value of 0 (thus, a migration event specified by the user is in fact split into two separate events- a start and end).

These events are communicated at run time by reading an input file (see event_table.tsv) of the following format:

```
#MCpopsim event definitions

###### Divergence times #######
#event  time     source  newpop          bottleneck

t       0        0       1       0.5
t       1000000          end     end

########### Gene flow #########
#event  start    end     source  sink    rate
m       900000           1000000         3       2       0.01

####### Hybridization ########
#event  time     pop1    prop1   pop2    prop2   newpop
h       100000           1       0.5     2       0.5     3
```

Here, event types are given with a lower-case letter, and the appropriate columns for that type are filled out. Order does not matter, as the appropriate data to collect is parsed by the event type in column 1, but it makes it more readable to organize it this way.

The workflow for a single generation is as follows:

1. Query event table with the current generation count
    - A. If an event occurs do the following:
        - a. "h" - call function to merge pops; push_back new merged pop to pop vector
        - b. "m" - alter migration matrix with new values
        - c. "t" - split populations; copy and push_back into pop vector
2. Permute by pseudosampling ALL alleles of all populations
    - A. Generate new frequencies
        - a. If new frequency is zero, delete from std::map
3. Generate new alleles by calling mutation function
    - A. mu is a probability; events occur by sampling a uniform dist from 0 to 1.
    - B. For a large mu, probability is decremented by random number and then the uniform is sampled again, thus multiple mutations are possible.
4. Correct population size if it has drifted from outside of coded bounds
    - A. new freq = freq(target size / current size)
5. parse Migration matrix (nested for loop; if non-0 value then exchange alleles between I and j
6. Generation count ++

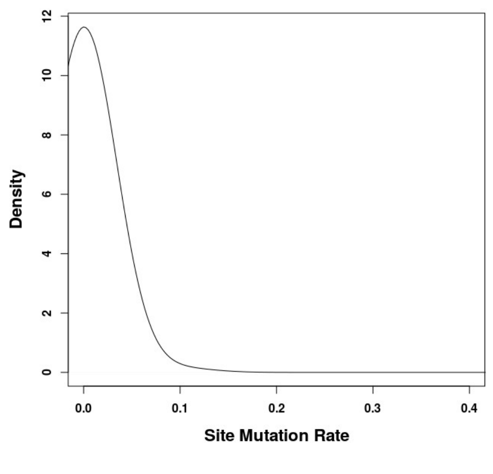
Outputs follow the FASTA convention (a population phylogenetic format), in two variations: one containing all alleles of the terminal simulation state, and another approximating a random sample of the population, where allele sequences are chosen as a weighted random sample, with the end-state frequency as the weight.

```
FASTA:
>Locus.1_Population.4_Individual.0_Allele.0(3)_Freq.1.000000
AAATCGATAAATATGCGACCACATCGATTAATAAACGAGGAGAAAGCGGT
…
```

I have also implemented a simple parallelization scheme with minimal inter-process communication (limited to an initial scatter of random number seeds, and some barriers to prevent a single process getting too far ahead). This was most easy to implement with process rank 0 generating a static event table which is shared by all other processes. Each process is given N/p loci to simulate, and each simulation is independent of others, and so we should approach a linear speed-up as the number of processes increases.

Simulation validation

Simulations yielded results conforming to expectations under population genetics theory. For sample of simulated loci, the average time until fixation of a new mutation, excluding those cases where the newly generated allele drifted to a frequency of 0, in a population of an effective size (N_e) of 1,000 individuals was 3,771 generations- a number which approximates the expectation of $4N_e$ generations under a strictly neutral model. Also following neutral expectations of a finite population, the probability of fixation of a new allele should be $1/2N_e$, and given an expected rate of novel alleles appearing of $2N\mu$, where μ is the per nucleotide mutation rate, the overall rate of fixation of novel alleles is equal to the mutation rate. In this case, the mutation rate was sampled from a Poisson distribution of mean $1e^{-6}$, and the observed probability of fixation of a mutant allele (i.e. derived after initiation of the simulation) was $6e^{-6}$, which deviates but not alarmingly so.



(Above): Sampled site mutation rates.