Internal *msms* Manual

March 9, 2010

1 Introduction

msms (ms mit[†] selection) or $(ms)^2$ is a program for simulation of different demographics and structured populations with selection, using a forward-backward approach. Generally, when finished it will support everything that ms does, but with selection at one locus.

One goal of the project is to produce code that is modular and accessible for extension by others. What this means in detail has not yet been decided. Currently, it means that the source and binary are always distributed together to allow for critiquing, changes and patch submissions.

The purpose of this manual is not to provide a user guide, but to provide details of the internals of the program. That is, to provide extra documentation over and above the source code in a more generally accessible manner. It should be noted that this documents what the code does, more than it discusses various assumptions of the underlying models. We may expand explanations to include this in the future. This document should be read before reading or altering the source code. Note this is still somewhat a β level document.

1.1 Simulation Outline

The simulation uses a two step method. First, we simulate forward in time, keeping track of the frequency of the beneficial allele using a standard Wright Fisher model with demographic structure. This means that the number of individuals in each successive generation is a binomial random variable with a parameter dependent on the previous generation. The details are given below.

Once the forward simulation is completed, we then use the coalescent to track sampled individuals back in time, conditional on the frequency of the beneficial allele deterimened from the forward simulation. This coalescent includes migration and recombination, and will also allow for many parameters to be functions of time. At this stage, we only assume that the beneficial allele frequency is a function of time. See the road map below for the expected order of added features.

[†]mit is "with" in German

The coalescent simulation is designed to be accurate within the normal coalescent approximation limits, even with quite extreme parameter choices. Furthermore, explicit events can be tracked on the tree if desired[†]. For example, we may wish to track when recurrent mutations occur. We can specify this at simulation time and get some summary statistics or tree labels on the events. However, if we are not so interested in such things, we incur almost no overhead when not tracking different events.

2 Forward Simulations

We consider a standard structured Fisher Wright population with two alleles a and A at a single locus. We assume that the allele A is beneficial and that we have a diploid population. The fitness values for the ith deme is $1+s_i^{aa}$, $1+s_i^{aA}$, $1+s_i^{AA}$ for genotype aa, aA, and AA respectively. Migration from deme j to deme i, is defined as the proportion m_{ij} of deme i that is made up of migrants from deme j. We let $m_{ii} = 1-\sum_j m_{ij}$, which is the proportion of non migrants in deme i.

Let $\boldsymbol{x}$ be a vector $(x_1, x_2, \ldots)$ where x_i is the relative frequency of the beneficial allele A. Therefore $x_i = n_i/(2N_i)$, where N_i is the population size of deme i, and n_i is the number of A copies in deme i.

The life cycle is selection, mutation, migration, and random mating. The census occurs with the formation of zygotes from the infinite gamete pool. Deviations from HWE within each deme due to sampling are ignored. Selection occurs at the zygotic stage and separately within demes. It may depend on the deme, however further extensions are not yet in place.

Now consider a single deme i with proportion x_i of the beneficial allele. From HWE, the amount of beneficial alleles after selection but before migration is,

$$\eta_i^A = x_i\left(1 + (1-x_i)s^{aA} + x_i s^{AA}\right) \tag{1}$$

and the amount of the non beneficial alleles is

$$\eta_i^a = (1-x_i)\left(1 + x_i s^{aA} + (1-x_i)s^{aa}\right). \tag{2}$$

We use the term *amount* to indicate that these are not proportions,ie are not normalized. Now if we consider the amount of beneficial allele in deme i that comes from the migrants from deme j we multiply the amount of beneficial alleles after selection in deme j by m_{ij}. If we consider all demes, including the non migrants from the m_{ii} term, the amount of beneficial alleles and non beneficial alleles after selection and migration are,

$$\eta_i^A = \sum_j m_{ij} x_j\left(1 + (1-x_j)s^{aA} + x_j s^{AA}\right) \tag{3}$$

$$\eta_i^a = \sum_j m_{ij}(1-x_j)\left(1 + x_j s^{aA} + (1-x_j)s^{aa}\right) \tag{4}$$

[†]this is not fully implemented yet

and we have redefined both η_i^A and η_i^a.

We can now write the new frequency x_i' in an infinite population by normalizing as follows,

$$x_i' = \frac{\eta_i^A + \mu\eta_i^a}{\eta_i^A + \eta_i^a} \tag{5}$$

where μ is the recurrent mutation rate. That is, the mutation rate from a to A, we consider the reverse mutation rate to be zero.

We now sample the "infinite" population with a binomial random variable and thereby include genetic drift. The number of A copies in the next generation n_i' is given by

$$\Pr(n_i'|n_i) = \binom{2N_i}{n_i'}(x_i')^{n_i'}(1-x_i')^{2N_i-n_i'}. \tag{6}$$

We then sample this using a modified library from the colt project[†]. If the beneficial allele disappears, the simulation is reset with a single allele, and the time reset to zero. This is repeated until the beneficial allele goes to fixation.

Haploid Populations

In the haploid case we only have two selection coefficients, $1+s^A$ and $1+s^a$. The formulas 3 and 4 simplify to

$$\eta_i^A = \sum_j m_{ij}x_j(1+s^A) \tag{7}$$

$$\eta_i^a = \sum_j m_{ij}(1-x_j)(1+s^a). \tag{8}$$

Equation 5 stays unchanged while equation 6 is changed to,

$$\Pr(n_i'|n_i) = \binom{N_i}{n_i'}(x_i')^{n_i'}(1-x_i')^{N_i-n_i'}. \tag{9}$$

3 Coalescent Simulations

Coalescent simulations are somewhat more involved due to recombination and selection. The coalescent rate is dependent on population size. However, the individuals that have the beneficial allele cannot coalesce with individuals that don't have the allele. Thus, the "virtual population size" for a lineage in say, the selected class, is a function of time. Further, it is not a well-behaved function of time since we include drift. This complication extends to most events. The events considered are migration, coalescent, recombination, and recurrent mutation.

[†] `http://acs.lbl.gov/~hoschek/colt/`

The simplest method is to use a generation by generation method. That is, we check in each generation if an event occurs with a probability that produces the correct geometric distribution where we define these probabilities later. This is the method used by the Yuseob Kim Selective sweep program, with interval rescaling to ensure that all events are sufficiently rare. This can be slow. However, we use an explicit N (ie we don't "scale" parameters, and $N = 10^6$ will mean that on the order of 10^6 tests need to be done), where the complexity scales with N, whereas with the standard coalescent it scales with sample size. This method was used originally, but is now replaced with the method described below.

3.1 Implementation Outline

When we are not in a selection phase, normal coalescent simulations are performed. That is, an exponential random variable is produced for a small set of events, then the shortest exponential variable is taken, and finally, the details of the event are decided; which often requires more random numbers to choose the specific event. This is reasonably well establised in the literature.

When we are in a selection phase, we first generate exponential random variables for recombination, the only event that is independent of allele frequencies, and take the first event. We then generate more exponential variables, using the fast coalescent method with the shortest current exponential time as a parameter. This way the fast simulations can return faster in many cases. Finally, we pick the event with the shortest time and "complete" it as above.

It should be noted that for most events we must generate an independent event for each deme and each allelic class due to the dependence on the allele frequencies. For example, we must generate a coalescent event time for each deme and each class, and pick the smallest result because the allele frequencies in each deme vary stochastically.

3.2 Fast Coalescent Simulations

The discrete method above produces geometric waiting times between events. We can approximate this with an exponential, with reasonably good accuracy; this is in fact what the original Kingman Coalescent does. So rather than just checking the probability of an event at generation t, we can produce an exponential random variable instead. However, we need to be able to take into account the changing rate of the process. Without considering a specific event, we just discuss the generic scheme used in the code. This is currently only used for coalescent events, but will be extended to other events when the appropriate parameters are time varying.

For concreteness and simplicity, we assume that we have a constant rate λ over a time interval $\mathrm{d}t$. The cumulative probability function of the exponential distribution is

$$\Pr(X \leq t) = 1 - e^{-\lambda t}. \tag{10}$$

So the probability of an event not happening in an interval of length $\mathrm{d}t$ is just the complement

$$\Pr(\mathrm{d}t < X) = e^{-\lambda \mathrm{d}t}. \tag{11}$$

Letting λ_i be the rate as a function of generations with $i \in \mathbb{N}$ and $i = \lfloor \frac{t}{\mathrm{d}t} \rfloor$. We assume that there is a constant part, and a variable part. So $\lambda_i = \lambda \alpha_i$ and α_i is now the function of generation and $\alpha_i > 0, \forall i$. The probability that an event does not happen after n generations is

$$\Pr(n < N) = \prod_i^n e^{-\lambda \alpha_i \mathrm{d}t} \tag{12}$$

$$= e^{-\lambda \mathrm{d}t \sum_i^n \alpha_i}. \tag{13}$$

This result can be used to generate exponentially distributed time intervals efficiently as follows. Let U be a uniform random variable on the interval $(0, 1]$. We can equate 13 with U and solve for n. We further simplify by setting $\mathrm{d}t = 1$ as t is in generations. We have

$$U = e^{-\lambda \sum_i^n \alpha_i} \tag{14}$$

$$\ln U = -\lambda \sum_i^n \alpha_i \tag{15}$$

$$\frac{-\ln U}{\lambda} = \sum_i^n \alpha_i. \tag{16}$$

We can then sum from the initial generation until the right hand side of 16 is larger than the the left hand side, and then use the previous n as an estimate. We in fact also allow for fractional "generations" by considering that we have a constant rate for the interval n to $n + 1$. Doing so allows rates to be higher than 1 without causing the program to stall[†]. Furthermore, the approximation in this case is still of the same order as the traditional coalescent.

In practice, we do not sum over n generations to solve 16. After generating the forward sweep, the relevant α_n parameters are stored as cumulants $\sum_i^n \alpha_i$, and a binary search is used. This gives a $O(\ln \mathcal{N})$ performance for generating this random variable, where $\mathcal{N}$ is the full range that the binary search must be performed over and is bounded by N. However, since we already generated some exponentially distributed waiting times, we can limit the search to a much smaller region, so generally $\mathcal{N} \ll N$.

3.3 Coalescent Events

Coalescent events can only happen between lineages in the same class and the same deme. Thus, there is a dependence on frequency of the selected allele as stated earlier.

[†]This can be the case with high recombination rates, or when selection is very strong

Let k_i^A be the number of lineages in deme i that is in the selected allelic class, ie with allele A. The coalescent rate between these lineages is

$$R_c^A = \frac{k_i^A(k_i^A - 1)}{4N_i x_i} \tag{17}$$

or

$$R_c^a = \frac{k_i^a(k_i^a - 1)}{4N_i(1 - x_i)} \tag{18}$$

for the unselected allelic class. Thus, when using the fast coalescent method, we set $\alpha_i = \frac{1}{x_i}$ or $\alpha_i = \frac{1}{1-x_i}$ as appropriate.

3.4 Migration

Migration when there is no selection is simple because the definition of m_{ij} is constructed to make it simple. Recall that m_{ij} is defined as the proportion of deme j that is made up of migrants from deme i. If we trace a lineage back in time, the probability that it migrates from deme j to deme i backwards in time is just m_{ij}. However, we need to consider the different allelic classes. That is, if a lineage migrates between demes, it cannot change allelic classes.

The total proportion of deme j that is in the selected allelic class is just x_j. While the total proportion that migrated from deme i is m_{ij}, and so the total proportion of migrated individuals that are in the selected allelic class is $\frac{x_i m_{ij}}{x_j}$. Hence, the migration rate from deme i to j in forward time[†] is

$$R_m^A = k_j^A \frac{x_i m_{ij}}{x_j} \tag{19}$$

$$R_m^a = k_j^a \frac{(1 - x_i)m_{ij}}{1 - x_j}. \tag{20}$$

Again we can now set $\alpha_i = \frac{x_i}{x_j}$ or $\alpha_i = \frac{1-x_i}{1-x_j}$ respectively for the selected class or unselected class.

3.5 Recurrent Mutation

Recurrent mutation also gains a dependence on the proportions of selected to unselected classes. However, since in forward time only individuals that are unselected can mutate to the selected class, we only have one case to consider. That is, in backwards time a lineage can only move from the selected class to the unselected class.

Now consider the proportion of the selected alleles in a deme that arises from mutation on the selected locus. This is simply $\mu(1 - x_i)/x_i$, that is the total amount of selected alleles that arise from mutation $\mu(1 - x_i)$ normalized by the total amount of selected alleles. So the rate backwards in time is

$$R_\mu = \frac{\mu(1 - x_i)}{x_i}. \tag{21}$$

[†]Thus in backward time the lineages moves from deme j to deme i.

3.6 Recombination

The recombination rate is not directly affected by the allele frequencies. However, there are a number of other considerations that make recombination the most difficult event to implement.

First, we must consider the sequence model. We use an abstract continuous model without considering an explicit mutation model at this stage. Here we only need to consider what sections of a sequence are *active*, in that we can observe the mutation process on that region from our sampled individuals. The sequence here is just a set of non overlapping intervals that denote these active regions. Furthermore, the regions are not constrained to have a summed length of 1, but can be smaller or larger. We define a single active interval at sampling time as $(0, 1)$. Multiple loci can be easily added by simply defining more than one active interval at sampling time.

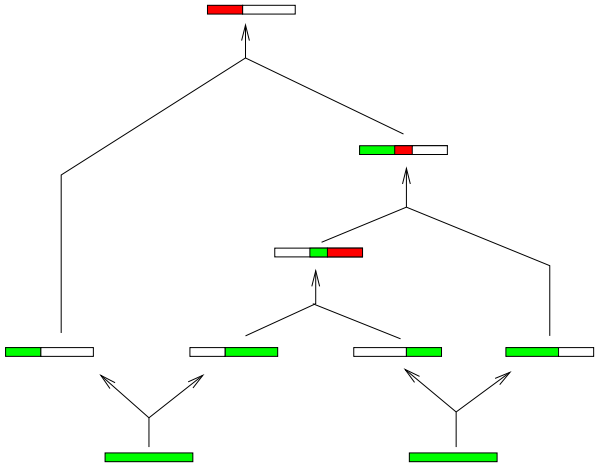

Figure 1: The figure shows how active regions (green) change with recombination events and coalescent events. Recombination events split active regions whereas coalescent events join them. However, some regions may "coalesce" (red). In this case, the region is active below the coalescent event but not active above the coalescent event.

Ignoring selection, we consider a simple coalescent recombination graph in figure 1. Here just two sequences have recombination events, and then the resultant 4 lineages coalesce. Going back in time, the recombination events increase the number of lineages present by splitting the active regions into complementary active regions.

When we have a coalescent event, we use a union of the active region of both coalescing lineages. However, we need to also consider *coalesced regions*. That is, active regions that are no longer present in any other lineage. Such regions are shown in red in Figure 1, and are not active above a coalescent event. But, they are active below the coalescent event. In other words, we cannot observe a

segregating site in our samples over the coalesced region from a mutation that occurs above[†] the coalescent event.

Over time, the total length of active regions reduces to zero, and we do not need to run the coalescent simulation to the ultimate common ancestor of the samples. Further optimization is needed when high recombination rates are used. It is a fact that some recombination events result in unobservable lineages. That is, a lineage with no active regions. This can happen if the cut site of a recombination event happens outside the upper or lower bounds of the active regions. Therefore, instead of using the recombination rate, we use the observable recombination rate for each lineage. This is the distance between the upper and lower bounds of active regions, times the recombination rate.

We note that at this stage we do not consider mutations above *coalesced regions*, and thus we are in fact considering polymorphisms. While in some cases, changes from the ancestral state are what we may be interested in. This will be added at a later time.

3.6.1 Recombination with Selection

When considering recombination with selection, we must take into account that there is a selected locus somewhere on the sequence. This locus can be either the selected allele, or the unselected allele. We do not explicitly tag sequences with the allele but rather track lineages as either in the selected class or unselected class. The locus has a position on the sequence, and can be some distance away from any active region. This has an impact on the observable recombination rate because recombination between this selected locus and active regions is observable. In other words, it changes the coalescent process.

When going forward in time, two different lineages are crossed over to form a new lineage. Here are the possibilities:

- Both lineages are from the selected class, and the resultant recombinant is also in the selected class.
- Both lineages are from the unselected class, and the resultant recombinant is also in the unselected class.
- One lineage is from the selected class and one from the unselected class. The resultant recombinant can be either in the selected or unselected class.

When moving backward in time during the coalescent phase, we start with the resultant recombinant. Let x be the relative frequency of the beneficial allele at the time of recombination in a single deme. We have only two cases to consider. If the recombinant is in the unselected class, then at least one source lineage is also in the unselected class. The second source lineage for a recombinant is from the selected class with probability x. Otherwise, it is from the unselected class.

[†]Above here means further into the past.

The complementary statement is true when the recombinant is from the selected class. At least one source lineage must be from the selected class. The second source lineage is from the selected class with probability x, and is from the unselected class otherwise.

Note that when the source lineages are from both classes, which lineage gets what active region is defined by the recombination cut site. That is, if the selected locus is on the left of the cut site and is selected, then the "left" lineage source must be from the selected class.

Finally, we consider a simple case that one of the source lineages has no active regions. This can be quite common when the selected locus is a long way from the sequence locus. A recombination event can be swapped for a simple change in class and hence, does not require an extra lineage to be tracked. That is, when going back in time a selected class lineage may move into an unselected class, as far as all active regions are concerned, or vice versa.

4 Random Numbers

The random number generator is not a default java random number generator, as this has a period of only 2^{48} and fails some random number tests in the die harder battery of random number tests. We have a fast random number generator that passes all die harder tests and avoids the hyperplanes problem, and has a period of $2^{128}-1$. The internal state 64 bit variable that is incremented in a m-sequence implemented via the *shift xor* method, combined with a 64 bit Weyl sequence. Intermediate output is via the addition of both sequences and masking. This is much faster[†] than mersenne twister, while still avoiding the LCG problems.

Note that other random number generators will be available in the future. Adding a new random number generator has been designed to be as easy as possible. Note that for some parameter choices, up to 50% of the CPU time is spent generating random numbers.

5 Current Status

As indicated above, there are a number of "missing" features. The code is still very much in a beta state. However, we foresee that over the next months, features will be added quickly.

5.1 Unit Tests

Currently, there are no unit tests in the source. They were cleared out when re-factoring because they were bigger than the base code, and contained some bugs. New unit tests will be added but will only test sub components to avoid unneeded complexity.

[†]about 10 times faster in fact

5.1.1 Correctness

The current code has been tested against Yuseob Kim's selective sweep program and against Hudson's MS program. However, using both migration and selection has not been directly compared to other programs as of yet. So testing with forward simulation programs will be undertaken shortly. Also, the tests have been done in an ad hoc manner. Full testing scripts will be added shortly.

5.2 Performance

The code at this time has not been optimized yet. We suspect there can be some significant gains in both speed and memory performance. However, there is no point going fast if you are wrong. So emphasis at this stage is on correctness.

Having said that, performance is competitive with both MS and selective sweep for a range of parameter choices. Generally rescaling parameters and using a smaller N will increase performance if a particualr parameter set is slow. High recombination rates will slow down simulation dramatically, regardless of other settings.

5.3 Road Map

The two big items are automatic tests, and full MS command line compatibility. At this stage, only some options are supported.

6 Contributing

Contact me directly for details. This will become more formal when the code moves out of beta. Currently, the source control is via `git` and so patch submission is probably the simplest way until the beta phase is completed.