ACG Users Guide

v. 1.0

September 2011

Brendan O'Fallon
Dept. of Genome Sciences
University of Washington

Contents

What *is* ACG?

ACG is a tool to infer population parameters and genealogical structure from genetic data. It estimates the shape of a likelihood surface (such as a Bayesian posterior) of a model given some data, and thus is a close relative of "genealogy samplers" such as Mr Bayes, BEAST, and LAMARC. In a typical analysis the model may include population size, recombination rate, the model of nucleotide evolution, and the ancestry of the samples described as an Ancestral Recombination Graph (ARG). ACG proposes small changes to the model, and accepts the changes with a probability that is an increasing function of the likelihood of the proposed model. Over time the model gradually evolves, and samples taken periodically (say, every 10,000 changes) can be treated as approximately independent samples drawn from the posterior.

ACG is written quite generally, and the user must pick which parameters, likelihoods, proposal kernels, and output options to use. Because of this freedom, ACG is not explicitly Bayesian, and priors need not be specified for every parameter. At this point it's up to the user to make sure that the analysis is constructed in a way that makes sense.

The primary design goal of ACG is to create a genealogy sampler that works with recombining sequences but had the same efficiency and simplicity as tree-only samplers. For several reasons samplers with recombination are significantly more complex than samplers without recombination, but our hope is to hide this as much as possible and provide an efficient, speedy search of ARG space.

A secondary goal of ACG is to provide a very modular and flexible environment to run MCMC analyses. To that end we've made it easy to install extensions developed by other folks. Just drop a .jar or .class file in the plugins folder and you'll be able to use tools that others have created. Similarly, the graphical interface has been designed to accommodate a wide variety of new models without needing significant changes.

Downloading and Installing

To download ACG, visit http://staff.washington.edu/brendano/acg , navigate to the "Downloads" page, and find the version (Mac, Linux, or PC) that matches your platform. After the file has downloaded, extract the contents either by double-clicking on the icon or right clicking and choosing "Extract...". After that, just click on the *acg* (or *acg.jar*) application to begin. You don't need to configure, "make", or install anything. On Unix-like systems (Mac & Linux) you can start ACG from the command line by typing:

```
$ java -jar acg.jar
```

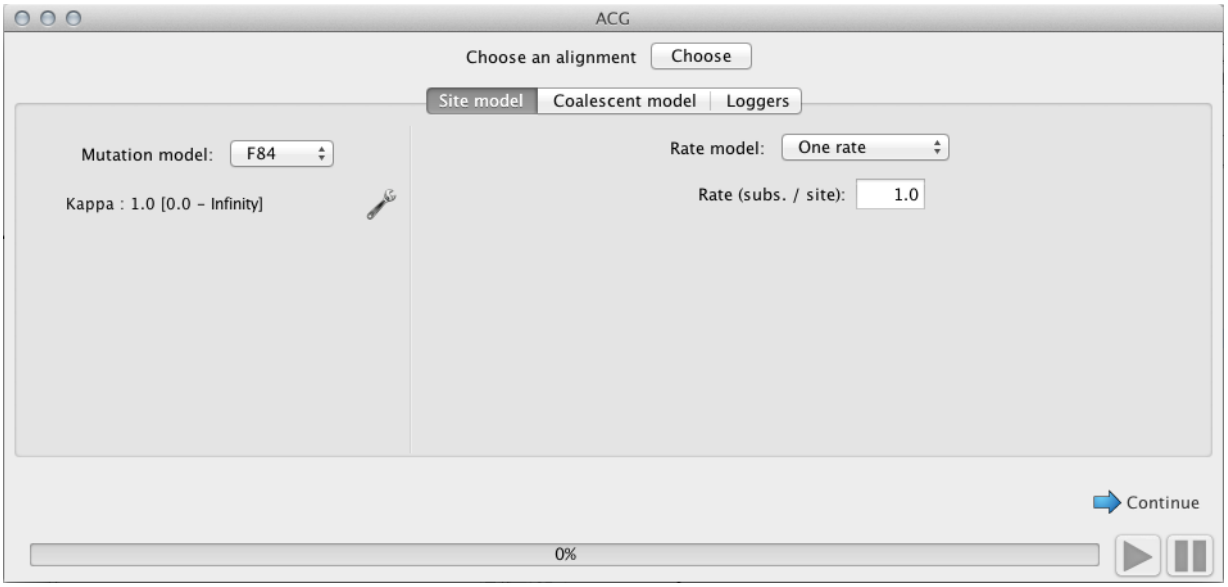
To run an input file called myInputFile.xml from the command line, just type

```
$ java -jar acg.jar myInputFile.xml
```

Quick start in 3 steps

1. Create the model

Start the application by double-clicking the *acg.jar* icon. Then click on button labelled "Create a new model". You should see a window that looks something like this:



You'll first need to pick an input alignment. To do this, just click the "Choose" button at the top of the window, and select a fasta or phylip formatted input file. When the alignment loads, you should see the text at the top of the screen change to something like "Alignment example : 20 sequences of length 10000".

You can now choose a few options for the evolutionary and coalescent model. Variants of different models can be selected from the lists, and the relevant parameters are displayed below. Many properties of individual parameters can be configured by clicking on the nearby wrench icon.

After you've selected the model parameters and priors, you'll probably want to pick a few "loggers" so you can gather data about the analysis. Individual loggers gather different types of data, the "StateLogger" is the most basic and produces a BEAST-style log file of parameter values and properties. Loggers can also collect properties like the root height over the length of the sequence, the positions of recombination breakpoints, and consensus trees at individual sites.

Finally, click the continue button to choose some options for the Markov chain itself. By default a single chain is run with no heating (for samples with long sequences and / or lots of recombination you'll probably want to turn the multiple chains option on). If you want to save these settings for future reference, click on the "Save" button at the bottom of the screen. Otherwise, just click "Run".

2. Begin the run

When you click the "Run" button the view will switch to a window that allows you to choose from many different parameters and options. By clicking on one of the check boxes, you indicate that you want to "watch" that parameter graphically during the run. *The run will proceed in exactly the same way no matter which boxes are checked.* Select a few of the parameters or likelihoods that you're most interested in, and finally click on the "Start" button to actually begin the run. After you choose "Start", the view will again switch and you'll see a bunch of panels, each one of which corresponds to one of the parameters that you've chosen. Many properties of these panels can be switched either by right clicking (or, on a Mac, Control-clicking) on a panel and selecting one of the various options, or by double-clicking on an element in the panel (for instance, a data series, axis, legend, or label).

When the progress bar at the bottom of the screen reaches 100%, you're done and it's time to examine the output.

3. Examine the data

The easiest way to examine output is to look at histograms of the parameter values. If you're interested in population size or growth rate (and you've run a model which estimates those parameters), histograms of the sampled values will approximate the likelihood distribution you're looking for. To view a simple histogram, just right-click on one of the panels on the output screen and choose "Switch to histogram". You can configure the number of bins in the histogram by again right-clicking and choosing the "Configure histogram" option.

For more advanced analyses, you'll want to examine the output of the various Loggers you've chosen. Different loggers generate different types of data, but all will store them in separate text files whose names you choose. Those most important of these is the "StateLogger", which records the values of the likelihoods and (some properties of) the parameters over time. The State Logger is optional, but recommended for every analysis since it's the only way to assess convergence of the chain. The State Logger produces a file whose name is specified in the input file, but usually ends with .log. This file can be opened in a program such as Tracer (free, available from http://tree.bio.ed.ac.uk/software/tracer) or in the statistical package R.

Other types of loggers produce their own output files, again with names that are specified in the input file. The format of these output files varies slightly from logger to logger but is pretty consistent. The Breakpoint Density logger produces a file that contains two columns, the first indicates the position along the sequence, the second is the relative density of recombination breakpoints (at any depth) at that position. For thorough description of the loggers, see the Appendix A.

Understanding ACG:

Background

At its core ACG does nothing more than run a Markov chain with Metropolis-Hastings (and reversible-jump) accept / reject steps. There's nothing explicitly Bayesian or even explicitly "genetic" about it, although we happen to provide a few tools that we hope are useful for genetic analysis. The basic distribution that ACG hopes to infer is

$$Pr\{ \text{Model} = x \mid \text{Data} \}$$

Put another way, how likely is some model that we've defined, given that a particular data set was observed? Most often, the Data will be a genetic sequence alignment, and the Model will be some description of population size, recombination, sequence evolution, and potentially many more parameters. Analytical evaluation of the expression above is nearly impossible for all but the simplest of genetic models. Nonetheless, one way to gather information about the distribution is to use Bayes' Rule to reverse the order of the conditioning and express the relationship in a Bayesian context :

$$Pr\{\text{Model} = x \mid \text{Data} \} = Pr\{ \text{Data} \mid \text{Model} \}\ Pr\{ \text{Model} \} / Pr\{ \text{Data} \}$$

The left hand side of the above equation has three parts. The $Pr\{ \text{Data} \mid \text{Model} \}$ is sometimes called the "evidence", and can often can be computed readily. The $Pr\{ \text{Model} \}$ bit is the "prior", it describes the probability of the model in the absence of the observed data. While often contentious, this part is typically easy to compute. The denominator, $Pr\{ \text{Data} \}$, is the probability of observed the data set we did given all possible models. This part is often impossible compute analytically. Fortunately, there's a way to avoid calculating the denominator involving Markov chain Monte Carlo. Yang & Rannala (1997) first described the use of MCMC for this type of phylogenetic inference, and more details of the method can be found there. Briefly, by comparing the relative likelihood of two different model states, the denominators (which do not depend on the model) cancel, allowing them to be ignored.

The end result of the procedure is not a single curve or distribution, but a collection of values sampled from the distribution. Because the Model has many parts, a "value" will be a collection of single instances of those parts – for instance a single ARG, a single estimate of the Ts / Tv ratio and population size, etc. Nonetheless, if all has gone well the mean of all the population size values (for instance) will be close to the true mean.

Description of the algorithm:

The input to the core algoirthm of ACG consists of a collection of Parameters, LikelihoodComponents ("Likelihoods"), and Modifiers. Every modifier is associated with exactly one Parameter. Parameters are associated with one or more Likelihoods, and each

Likelihood is associated with one or more Parameters. Parameters encapsulate a state that may be adjusted, and the state is defined very generally. It may consist of a single value (for instance, population size), a collection of values (for instance, the base frequencies), or more complex structures (such as an entire ARG). Likelihoods compute a single likelihood score that depends on the states of the Parameters. For instance, a Likelihood may compute the probability that the population size is equal to 0.01, or that the given genetic alignment was observed conditional on an ARG. Together, the Parameters and Likelihoods are taken to define a "Model", and ACG purports to identify the likelihood distribution of Parameter values given some observed data.

To accomplish this, a Markov chain is constructed whose stationary distribution is the desired probability distribution (*Pr*{ Model | Data }). States in the chain consist of exact values of the parameters. New states are generated according to the Metropolis-Hastings-Green algorithm, described below:

1. A Modifier is selected by first choosing a Parameter in proportion to the value of its "frequency" attribute, then choosing a Modifier connected to that Parameter in proportion to its "frequency" attribute relative to the frequencies of the other Modifiers associated with the Parameter
2. The chosen Modifier proposes a new value for the associated Parameter and computes the Hastings ratio associated with the given proposal.
3. The likelihood of the new state, L^*, is computed by querying all LikelihoodComponents.
4. The acceptance probability of the new state, say , is computed as =Min(1, $L^* / L^* H$), where L^* is the probability of the proposed state, L is the probability of the current state, and H is the Hastings (or Green, in the case of reversible-jump moves) ratio of the move.
5. If a uniform random variate r has value < , the proposed state is accepted. Otherwise, the new state is equivalent to the current state.

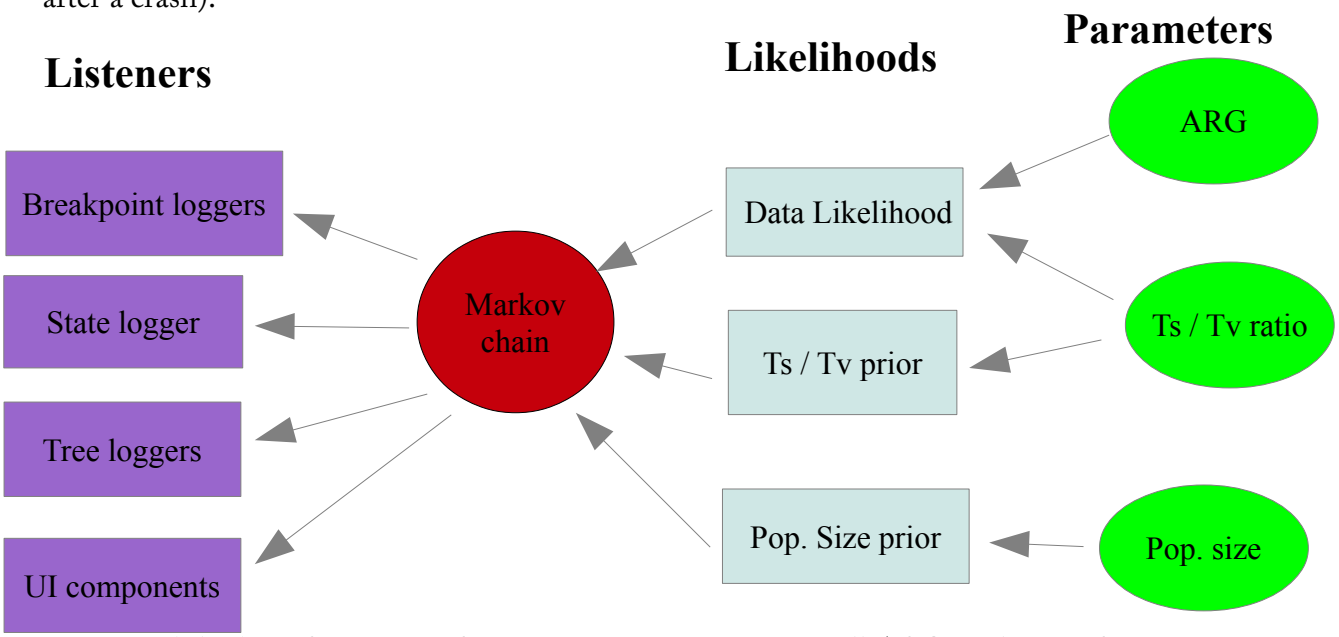
The Structure of ACG

There are four main types of objects in ACG: Parameters, Likelihoods, Modifiers and Listeners. As mentioned above, a Parameter is any object that can influence a likelihood. In a typical genetic analysis a Parameter may be something like the transition to transversion ratio, the population size, the set of base frequencies, even the entire set of ancestral relationships of the sequences (the ARG). Likelihood objects are anything that can compute a likelihood that depends on the state of one or more Parameters. For instance, the probability of the ARG conditional of the data is a likelihood component (the "Data Likelihood"), and so is the probability of an ARG given the population size and recombination rate. When a new value for a Parameter is proposed by ACG, the various Likelihoods that depend on that Parameter are informed that they need to compute a new value.

A Modifier is a type of object that can propose a new value for a Parameter (these are often called "proposal kernels"). Different Modifiers are appropriate for different types of

Parameters. You can't modify an ARG in the same way that you modify the population size, for example.

Finally, Listeners wait around and "listen" for changes in the Markov chain, and periodically do something. Most Listeners write information to files, and these are called "Loggers". Loggers collect information about the state of the chain, for instance the values of the Likelihoods and Parameters, and add it to files that can be examined. See below for a complete list of loggers. Other types of Listeners have to do with debugging or utilities (for instance, writing the current state of the ARG so that MCMC state can be partially recovered after a crash).

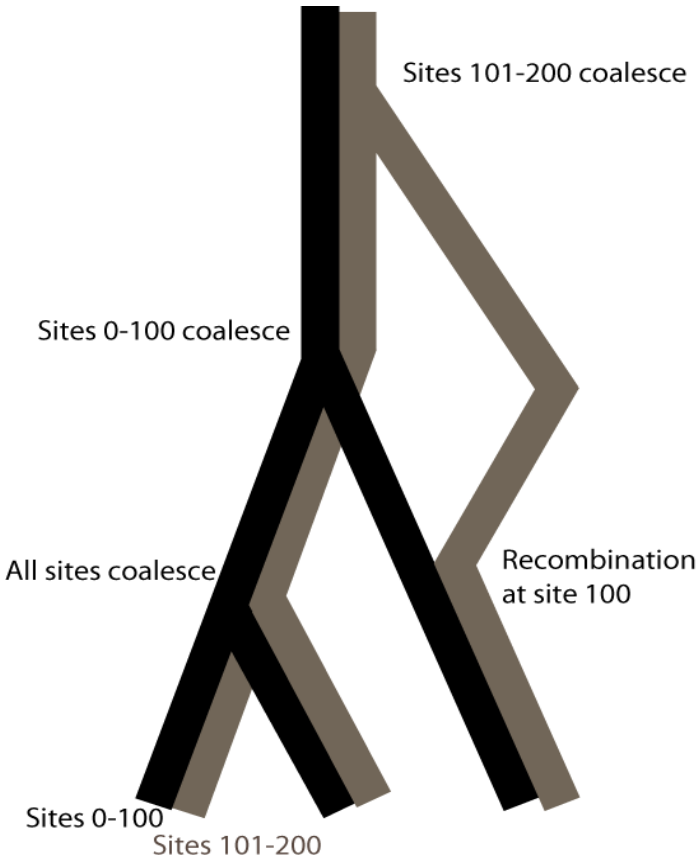


While these four types of components are common to all ACG analyses, a few extra types provide some additional functionality. For instance, a Sequence object stores a sequence, and an Alignment object stores a list of sequences (or a reference to an input file). These types of objects are sometimes required to create a Parameter or Likelihood – for instance, when creating a DataLikelihood object, you'll need a reference to an Alignment as well as an ARG.

An important note about priors: As mentioned above, there's nothing explicitly Bayesian about ACG. It doesn't "know" the difference between the prior and "evidence" portions of the posterior likelihood, and it doesn't require that prior distributions are defined for all parameters. It simply computes the likelihood of whatever components you've defined, and then accepts or rejects the modifications that are proposed. If you don't specify a prior for a parameter, then all values of that parameter are equally likely. Because of this, the burden is *on the user to ensure a that priors are defined in a way that makes sense.* ACG comes with a few kinds of priors built in, including uniform, exponential and gamma distributions (these are defined in priors.UniformPrior, priors.ExponentialPrior and priors.GammaPrior), and these priors are simply likelihood components that compute the likelihood that an associated parameter has a given value.

Understanding ARGs

Ancestral Recombination Graphs (ARGs) are an essential component of most ACG analysis. They're much like a simple genealogy of tree, but they allow for recombinations in addition to coalescences of lineages. In an ARG, lineages can branch as they go both forward (toward the tips) like a tree, and also backward (toward the root) in time. Splits in backwards time represent recombinations, and may significantly alter the topology of the tree. With



recombination, site 1 in an alignment may have a different ancestral tree than site 2. We refer to the tree that is ancestral to a certain site as a "marginal tree". ARGs with more than a few recombinations are difficult to draw and visualize, thus we often resort to dividing an ARG up into its constituent marginal trees and examining them separately (remembering that they're not independent for statistical analysis).

Confusingly, while some recombinations may have a large effect on tree topology, many, if not most, will not. Recombinations may be "trivial" (also called "cryptic"), such that they do not affect the tree shape at any site in any way. Such recombinations are not in principle detectable from genetic data. Other types of recombinations may only affect tree shape slightly, for instance by changing the time of coalescence for some sites by a small amount. These recombinations will be very difficult to detect unless the data has is very informative. ACG hopes to identify as many recombinations as possible, but since so many leave little or no trace

in the data we should abandon any hope of pinpointing the exact ARG.

An additional confusing feature of ARGs is that not all features are "visible" from the tips. Some branches and some recombination points are simply not accessible if one traces a particular site from the tips toward the root. For instance, a recombination breakpoint at site 5 is not visible on a branch that is ancestral to sites 10-20. Thus the recombination cannot in principle have any effect on the likelihood of the data given the ARG. Nonetheless, these features are still part of the analysis. Since a breakpoint may *become* ancestral due to rearrangements on other parts of the tree, these "invisible" or "inaccessible" features are maintained at all times in the ARG structure. Consideration of such features is of critical importance for an accurate sampling of ARGs from the distribution defined by the population size and recombination rate.

Many properties of ARGs are viewable from the main application window, under the ARG section of the panel that allows you to choose with properties to log. A brief explanation of each of these follows:

total.recombs : The total number of recombination breakpoints in the ARG

visible.recombs : The number of recombinations that are ancestral to sites observed in the tips. This may be substantially less than the total number of recombinations.

v**isible.height** : The height of the deepest coalescent node in the ARG *at which observed sites coalescence.* For simpler ARGs this will often be the total depth of the ARG, but for more complicated ARGs all sites may end of having a common ancestor at nodes tipward of the root. Another way of thinking about this is that it's the deepest marginal TMRCA for all sites.

num.nodes : Counts the number of contiguous ranges of coalescing sites, over which the data-likelihood calculating machinery operates. Often the speed of the chain is directly proportional to this value. For simple trees this is always (number of tips) -1 , but the number grows rapidly with recombinations.

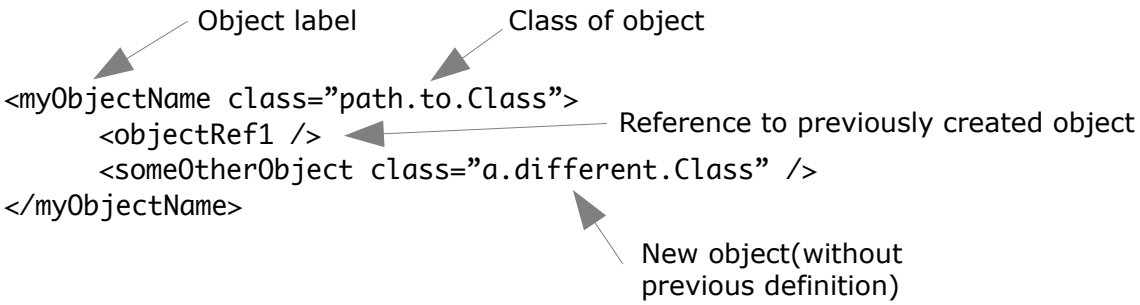
num.patterns : A pattern is the fundamental unit of data likelihood computation – it's a single site at which the data likelihood is computed. Since ACG attempts to "compress" or "alias" sites with the same data pattern as much as possible this number is far less than total number of sites * total number of coalescent nodes.

Anatomy of an input file

All ACG analyses are defined by a text-based input file that contains both the data as well as the type of analysis to be performed. When you use the GUI to create a model, it creates one of these input files behind the scenes. And while the GUI works for many models, more complicated models can be constructed by altering the XML itself.

Input files can be a bit confusing at first, one of the best ways to get acquainted with them is to look at the example input files in the doc/ folder. In this section we go through a few of the basics.

An input file is written in XML and begins with an `<ACG>` tag. Each element of the XML file corresponds to an "object" used in the analysis. The first time a reference to an object appears the object is created, and later XML nodes with the same name are assumed to refer to the same object. Objects are created with a tag that looks something like:



"myObjectName" is an arbitrary label for the object. It can be anything you like, and is just there so you can refer to the same object again later. The class="some.Class" part defines what *type* of object this is, and is required for the first reference to an object. The *class* attribute is case sensitive, so be sure to get it right. For later references to the same object you can just use `<myObjectName />`, and skip the class part – the class is only required when you create the object.

Some objects require references to other objects in order to be created. For instance, the DataLikelihood object requires a reference to both an ARG as well as a mutation model. This is accomplished with the following syntax:

```
<dataLikelihood class="dlCalculation.DataLikelihood">
	<myMutationModel />
	<myARG />
</dataLikelihood>
```

Note that order is important here, you'll get an error if you reverse the order of the arg and mutation model references.

Some objects require references to be in "list" form. This typically happens where an object can't anticipate the number of other objects it will get references to. For instance, an alignment

doesn't know in advance how many sequences it will be given. In this case, the sequences must be wrapped in a list, like this:

```
<alignment class="sequence.Alignment">
      <sequences class="list">
            <sequenceOne class="sequence.Sequence">
                  ATAGGATACGACGACTAGCGACTACG...
            </sequenceOne>
            <sequenceTwo class="sequence.Sequence">
                  ATAGGATACGACGACTAGCGACTACG...
            </sequenceTwo>
            ….
      </sequences>
</alignment>
```

The special class="list" attribute creates a variable-length list of objects. Other cases where you'll need to make a list include passing modifiers to Parameters and providing lists of Likelihoods, Parameters, and Loggers to MCMC objects.

Much more about what's legal and what's not legal in an ACG input file can be learned by examining the example input files, usually located in the doc/ directory. These are relatively well commented and progress from simple to more complex, hopefully giving you a chance to learn as you go along.

Walkthrough of a simple example

To begin, we'll start with the input file called `example1.xml`. It contains simulated data from 10 sequences of 2000 nucleotides each. The goal will be to infer a few basic properties such as the (scaled) population size, the shape of the genealogy, and the locations of recombination breakpoints.

Start by "loading" the input file in ACG. To do this, choose the "Load an input file" option from the first screen in ACG, then navigate to the "`doc`" folder, and choose the file called "`example1.xml`". When you select the file, the view will switch to show you an overview of the model, including which options were chosen and the associated parameters and their properties. Additionally, you can configure / select new loggers from the "Loggers" tab.

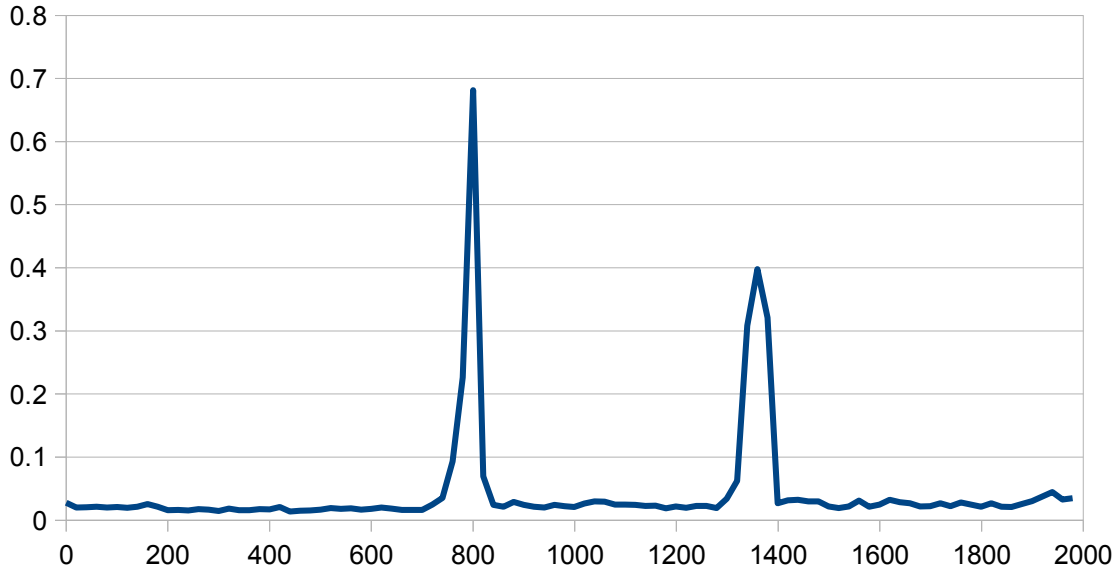
To begin the actual run, click on "Continue" and then the "Run" button in the lower right hand part of the window. You'll now be confronted with a potentially confusing array of parameters, parameter properties, likelihoods, and some utility functions. This screen allows you to choose which items you want to "watch" during the run (the run will proceed in the same way no matter which options you choose here). Since we're interested in population size, choose that one. It's also often a good idea to select "Data Likelihood", which is the probability of the data given the ARG. The Data Likelihood is often one of the last ones to reach convergence, hence if the sampling looks good for it you should feel somewhat confident that the sampling will be OK for other parameters as well. Other parameters of interest often include "visible.recombs" (the number of recombinations accessible from the data) and "visible.height" (the deepest point in the ARG at which a Most Recent Common Ancestor is reached). After you've selected a few of these, click "Start" to begin the actual run.

Depending on how fast your system is it will take a few (4-5) minutes for the run to complete. Examination of the traces in the GUI should help you to assess convergence (whether the chain has been run long enough to sample the parameters adequately). In addition, many basic properties of the analysis can be read directly off of the output screen, depending on which parameters you have chosen. Switching to "Histogram mode" (by right-clicking or control-clicking on the chart and then choosing "Switch to histogram") can also be a quick way to examine the distribution of values of a parameter.

While the GUI gives can give you a rough overview of the parameters and likelihoods, a more thorough examine requires looking at the files the loggers have produced. The State Logger should have created a file called example1.log. This file can easily be imported examined with tools such as R, Tracer, or a spreadsheet program such as OpenOffice (or LibreOffice) Calc. As mentioned above, one of the first columns to examine is "visible.recombs", which records how many recombination breakpoints were ancestral to the your data. In this case, you should see a mean number of breakpoints at around 4-6. (The total.bps column records the total number of recombination breakpoints. Not all of these will be ancestral to the data, and hence this statistic is somewhat less interesting than sites.bps.) The fact that the mean number of recombinations is relatively high suggests that there were

important recombination breakpoints in our data. Similarly, the distribution of the recombination rate parameter does not include zero, again demonstrating the importance of recombination in the data.

But where are these recombinations? Try opening the example1_bp.txt file, and viewing the contents in R or a spreadsheet program. It should look something like this:



The x-axis denotes position along the sequence, and the y-axis denotes the density of recombination breakpoints. Thus, the analysis feels confident that there's a recombination sitting near site 800 and another near site 1350, but little evidence for recombinations elsewhere.

At this point you might be a little confused; didn't the state logger say that there were around 4-6 recombinations? Where are the rest of them? The answer to this question lies in the fact that the MCMC is constantly exploring new ARGs, many of them with additional recombinations. The time the chains spends exploring a certain part of ARG space is proportional to the likelihood of that ARG, so if a recombination makes the ARG a lot more likely, the chain will spend lots of time with that recombination in place. However, many recombinations won't have much of an impact on the likelihood, since they're trivial, or nearly so. These will come and go at all points along the length of the sequence with equal probability, and hence don't show up as a single peak.

What to do now? Well, you've gathered strong statistical support for recombinations breakpoint near site 800 and 1400. You may be interested in what the trees look like on either side of those breakpoints. To investigate this, you could re-run the analysis, this time with loggers that collect the consensus trees on either side of the breakpoint – for instance, at sites 400 and 1200. Each consensus tree is a constructed from the marginal trees sampled at individual sites, and is a 50% majority-rule consensus tree, indicating that any clade present in over 50% of the sampled trees appears in the consensus tree, while unresolved clades appear as polytomies. By taking a close look at the consensus trees, you'll begin to get an idea for what the structure is on either side of the breakpoints.

Using the command line

In addition to the graphical user interface (GUI), ACG may also be executed from the command line. It's a java program, so execute it with the following command

```
$ java -jar acg.jar [inputFile.xml]
```

where [inputFile.xml] is the name of the file you wish to execute (if you do not supply an input file, the GUI will open). For runs with lots of sequences or MC3 runs with lots of chains , you may wish to increase the memory allotted to acg. To do this, try the following command

```
$ java -Xms1024m -Xmx1024m -jar acg.jar [inputFile.xml]
```

This increases the amount of RAM given to acg to 1.0G, which is sufficient for most runs (you can always increase it to more if you wish, I believe 4 GB is the maximum allowable value).

An important note: By default, the state logger does not write its output to standard out, and thus executing an input file you've made with the GUI will result in very little useful information being written to the screen. If you want output from the state logger to be written to the screen (for a more BEAST-like experience), add `echoToScreen="true"` to the state logger object in the input file.

If you still don't see any output, it may be that the Markov chain has been set to not run immediately upon creation. Be sure the Markov chain object (the one with `class="mcmc.MCMC"`) has the `run="true"` attribute.

Using argutils.jar

A handful of utilities are provided in the ancillary argutils.jar file. It can be executed from the command line by

```
$ java -jar argutils.jar
```

although doing so without any options or files specified will result in a brief help message. Right now the following options are supported:

--extract-trees [argfile.xml] : Extract all marginal trees from a given arg file as newick strings. The format is similar to ms, and can be read by programs such as Seq-Gen

--extract-tree [argfile.xml] [site] : Extract the marginal tree at a given site as a newick string

--emit-bps [argfile.xml] : Emit the locations (site & height) of all recombination breakpoints

--emit-tmrca [argfile.xml] : Emit the time to most recent common ancestor along the sequence

--scale [factor] [argfile.xml] : Emit a copy of the arg with all branch lengths scaled by the given value

--genarg [tips] [theta] [rho] [sites] : Generate a random arg with the given number of tips, theta, recombination rate (rho), and number of sites, and emit it as an xml file.

--convert [sequencefile.fas | sequencefile.phy] : Generate an ACG-style alignment block from the given fasta or phylip-formatted sequence file

--consense [treeLogFile.txt] : Generate a majority-rule consensus tree from the list of trees in a tree log file

Appendix A: Loggers

Loggers provide a flexible way of collecting useful information from a running Markov Chain. They sample the chain at a certain "frequency", typically every 1000-10,000 states, and store information related to parameter values and properties. To the extent possible loggers use a common attribute naming scheme, as follows :

burnin = "X" : Exclude the first X states from data collection
frequency="X" : Sample the Markov chain every X states
filename="X" : Write collected data to a file with the given name
site="X" : For MarginalTreeLogger and ConsensusTreeLogger, sample trees at the given site index

State Logger [logging.StateLogger]: The state logger is the most basic type of logger. It asks the Markov chain for all Likelihood and Parameter components, and writes their values periodically to a file that you specify. Each Parameter may choose to define 'value' in a different way. For instance, the transition to transversion ratio in the F84 model has a pretty clearly defined single number as a value. The Base Frequencies parameter, however, has four separate values. ARGs are also a Parameter, but can't easily by summarized by any number. For ARGs, several different characteristics are reported to the state logger. An additional feature of the State Logger is that it tracks the speed of the analysis, and reports it in MCMC states per second. Because the State logger queries the Markov chain for the list of components and parameters, you don't have to supply it with any parameters for it to function.

Marginal Tree logger [logging.MarginalTreeLogger]**:** The marginal tree logger writes out the (non-recombining) single tree at a particular site periodically to a file. The list of trees can then used to build a consensus tree to summarize the ancestral information at a single site, or to examine for other properties.

ConsensusTreeLogger [logging.ConsensusTreeLogger] : This logger builds a majority-rule consensus tree out of the trees sampled at a given site. The tree is formatted as a newick string, and clades in the tree are that present in at least 50% of the marginal trees sampled by the MCMC. Tree nodes are 'annotated' with two values enclosed as key=value pairs within square brackets, with *support=X* describing the fraction of trees containing the clade descending from the given node, and *error=Y* describing the standard deviation of node heights among all nodes containing the clade.

MPE ARG logger [logging.MPEARG]: Writes the full ARG with the maximum probability given the data to a file.

Last ARG logger [logging.LastARGLogger]: Primarily a utility, this just writes whatever the current ARG is to a file (overwriting the previous ARG), by default every 10000

MCMC states. This is often convenient so that MCMC runs can be continued from a point near where another run left off.

Marginal tree height logger [logging.RootHeightDensity]: Collects posterior distribution of TMRCA at each site across the length of the sequence. By default, seven values are reported, the lower 95%, 90% and 80% HPD boundary, the median, and the upper 80%, 90%, and 95% HPD boundaries. In the same file, the mean tree height across sites is also written.

Breakpoint density logger [logging.BreakpointDensity]: Builds a histogram over sites of the location of recombination breakpoints that are ancestral to the data (recombination breakpoints that are not ancestral to any sites are ignored). By default, the sequence is divided into 500 bins.

Breakpoint location logger [logging.BreakpointLocation]: Builds a 2D histogram of the locations of breakpoints, where one axis extends across sites in the sequence, and the other axis extends across tree heights. This enables one to pinpoint the location of a recombination not only along the sequence, but also in time. The densities of breakpoints (only the ones that are ancestral to the data) are stored as single real numbers in a large matrix. The easiest way to view the matrix is to import it into R (or a similar program), and use the image(x) command, where x is the matrix of values. This will produce a "heat-map" style plot of the areas where recombinations are likely to have occurred.