# Idiot's Guide to BEAST: A Manual for the Computer-Incompetent

***A companion for "A Rough Guide to BEAST 1.4," (July 2007) available at***

***http://beast.bio.ed.ac.uk/#A_Rough_Guide_to_BEAST_1.4***

*("Idiot's Guide" based on version 1.4.7, downloaded April 2008)*

Sara Volk, UTMB, rev. Nov 19, 2008

The Cliffnotes:

1) Make a nexus file from your alignment (coding sequence only?)
2) Upload .nex file into BEAUti; define all priors (dates, clocks, tree/demographic priors), assign run lengths, and generate a BEAST file (.xml file)
3) Upload .xml into BEAST. Run file until it's finished (it automatically saves trees and logs MCMC results); copy or otherwise make note of tuning optimization suggestions at the end of the run.
4) After run, open Tracer and load the .log file. Examine the means, distributions, and ESS values for parameters of interest. Determine whether your run was long enough to appropriately sample your parameters of interest.
5) (IF NOT LONG ENOUGH:) Rerun, either using the exact same .xml or possibly an operator-optimized prior (MUST HAVE THE EXACT SAME PRIORS, MODELS, AND TAXA!). Combine the log files using LogCombiner; open the combo log in Tracer to examine your parameters of interest.

More detailed instructions:

## MAKING THE NEXUS FILE

Use your preferred program to manipulate your alignment and create the .nex file (*e.g.*, MacClade, Mesquite). It is useful to have your sequences at least start in frame. If they do not, it is possible to "partition" your alignment into coding and non-coding chunks; then, BEAST will appropriately assign codon position (some nucleotide models can take into account difference in rate of variation based on codon position). However, this is more complicated, since it cannot be done in BEAUti (you have to manually edit the .xml file; see the Manual, pg 30).

TAXON NAMES: If your samples have different collection dates, you will find it useful to standardize the location of the date reference in your taxon names; it's most useful to simply use all four digits of the year (e.g., USA1995). You can incorporate month, if you wish -- see "Dating," under "BEAUti". If some of your samples are missing date information, you have two options:

1) Exclude them (you can delete them from your alignment either here or in BEAUti, if you want.)
2) Make an educated guess (this could significantly affect your results).

Do NOT leave the sample undated, as BEAST will interpret that as "current/this year."

## BEAUti: used to make the BEAST file

Start BEAUti, then open your .nex file. If you had a template, you would apply it now (file → apply template). In the absence of a template, you need to define dates for each strain, then define various priors for the alignment. A useful tutorial can be found at: http://beast.bio.ed.ac.uk/Tutorial_1.

**Data:** The default date assigned by BEAST is 0, which is appropriate if all of the samples were collected at the same time (relatively). Otherwise, you can add date information manually to each strain. If the date has been incorporated into the taxon name, you can click "guess dates," and it will hopefully fill in the appropriate dates. At the top of the window you can set the units that the dates are given in (years, months, days) and whether they are specified relative to a point in the past (as would be the case for years such as 1984) or backwards in time from the present. I wanted to include month information, so I decided: 1) all samples without collection month information would be designated with ".5" (equivalent to July); 2) otherwise, they are assigned as follows:

- January: (0)
- February: (1/12 = .083)
- March: (2/12 = .167)
- April: (3/12 = .25)
- May: (4/12 = .333)
- June: (5/12 = .417)
- July: (6/12 = .5)
- August: (7/12 = .583)
- September: (8/12 = .667)
- October: (9/12 = .75)
- November: (10/12 = .833)
- December: (11/12 = .917)

For example, a sample collected in May of 1995 would be given a date of 1995.333.

**Taxa:** If you want determine the date of divergence for any nodes, this is where you define taxon sets to do so. If you're only interested in rates, then you can ignore this tab. Note that each taxon set you define will cause the run to be a little slower. To create a taxon set, click the "+." All the taxa will appear in the "excluded taxa" – move the members of whichever lineage you'd like to date to "included taxa." Double-click the taxon set name to rename. To date more than one node/lineage, just add more taxon sets. If you want to constrain all of the members of the set to be monophyletic, click the box .

**Model:** This allows you to set any priors on nucleotide substitution model and rate, and clock-ness. Most of the info from this section has been stolen from the BEAST manual; I added as appropriate.

- Use SRD06 Model: When this button is pressed BEAUti selects a particular combination of settings which represent the model suggested by Shapiro et al [**Shapiro** B., Rambaut A., Drummond A.J. (2006) Choosing appropriate substitution models for the phylogenetic analysis of protein-coding sequences. Mol. Biol. Evol. 23(1):7-9.]. This model links 1st and 2nd codon positions but allows the 3rd positions to have a different relative rate of substitution, transition-

transversion ratio and gamma-distributed rate heterogeneity. This model has fewer parameters than GTR +gamma + invariant sites but has been found to provide a better fit for protein coding nucleotide data. (that's why I used it).

- Substitution Model: Substitution models describe the process of one nucleotide or amino acid being substituted for another. There are two DNA substitution models available in BEAUti: the Hasegawa-Kishino-Yano (HKY) model and the General Time Reversible (GTR) model. Other substitution models can be achieved by editing the BEAST XML file generated by BEAUti. For nucleotide data, all the models that are nested within the GTR model (including the well known HKY85 model) can be specified by manually editing the XML. When analyzing protein coding data the Goldman and Yang model can be used to model codon evolution [Goldman N, Yang Z: A codon-based model of nucleotide substitution for protein-coding DNA sequences. Mol Biol Evol 1994, 11(5):725-736.]. For the analysis of amino acid data the following replacement models can be used: Blosum62, CPREV, Dayhoff, JTT, MTREV and WAG (ref).
- Site Heterogeneity Model: This allows the refinement of the HKY or GTR model to allow different sites in the alignment to evolve at different rates. The "None", "Gamma", "Invariant Sites" and "Gamma + Invariant Sites" options in this menu help explain among site rate heterogeneity within your data. Selecting "None" specifies a model in which all sites are assumed to evolve at the same rate. For most data sets, this will not be the case, however for some alignments there is very little variation and the equal rates across sites model can't be rejected. Selecting "Gamma" will permit substitution rate variation among sites within your data (i.e., the substitution rate is allowed to vary so that some sites evolve slowly and some quickly). The shape parameter "alpha" of the Gamma distribution specifies the range of the rate variation among sites. Small alpha values (< 1) result in L shaped distributions, indicating that your data has extreme rate variation such that most sites are invariable but a few sites have high substitution rates. High alpha values result in a bell shaped curve, indicating that there is little rate variation from site to site in your sequence alignment. When alpha reaches infinity, all sites have the same substitution rate (i.e., equivalent to "None"). If the analysis concerns protein coding DNA sequences, the estimated gamma distribution will generally be L-shaped. If the codons are however partitioned into 1st, 2nd and 3rd positions, 1st and 2nd will generally have a lower alpha value than the 3rd. Selecting "Invariant Sites" specifies a model in which some sites in your data never undergo any evolutionary change while the rest evolve at the same rate. The parameter introduced by this option is the proportion of invariant sites within your data. The starting value of this parameter must be less than 1.0, or BEAST will fail to run. Finally, selecting "Gamma and Invariant Sites" will combine the two simpler models of among-site rate heterogeneity so that there will be a proportion of invariant sites and the rates of the remaining sites are assumed to be Γ-distributed. You can also choose the number of categories for discrete approximation of the gamma distribution.
- Partitioning into codon positions: BEAST provides the ability to analyze multiple data partitions simultaneously which share or have separate parameters for each partition. If the analysis concerns just non-coding DNA like mtDNA control region, select "Off" from the menu. Partitioning is useful when combining multiple genes (e.g., cyt b and COI), protein and non-

coding sequence data (control region and cytochrome b), and nuclear and mitochondrial data, or to allocate different evolutionary processes to different regions of a sequence alignment like codon positions. By partitioning your data, this allows more information from the data set to be extracted. In BEAUti, you can only partition into 1st, 2nd and 3rd codon positions. All other partitioning must be done by editing the XML file. There are two choices. Partitioning into "(1+2)+3" keeps the 1st and 2nd positions in one partition (slower substitution rate due to their constrained nature in coding for amino acids) and the 3rd position in a separate partition (faster substitution rate due to the increased redundancy in the genetic code of the 3rd codon position). Partitioning into "1+2+3" allows each codon position to have its own substitution rate. This assumes that the data is aligned on codon boundaries, so that every third site in the alignment is the third position in a codon for all sequences in the alignment. Unlinking substitution model across codon positions will instruct BEAST to estimate a separate transition-transversion ratio (kappa parameter) for HKY or separate relative rate parameters for GTR for each codon position. Unlinking rate heterogeneity model will instruct BEAST to estimate the among-site rate heterogeneity parameters independently for each codon position.

- Fix mean substitution rate: UNCHECK THIS BOX! The only time we'd want to use this is if we just wanted to make a tree disregarding the time-stamps of the strains. Setting this to 1.0 will result in the ages of internal nodes being estimated in units of substitution/site, which is often appropriate when the objective of the analysis is phylogenetic reconstruction and the time frame of the phylogeny is not of interest. Otherwise, this option is relevant when you have no fossil calibration data and want to calibrate the data set using a known substitution/mutation rate. This will in effect calibrate the phylogeny with an external rate and will mean abandoning any errors associated with this rate.
- Molecular clock rate variation model: This allows you to select the appropriate model for rate variation among branches in the tree. The model you select will be used to estimate the substitution rate for each node of the tree, the tMRCA of taxon groups, and the treeModel.rootHeight parameter (which represents the tMRCA for the root of the phylogeny). There are currently three options in BEAUti: "Strict Clock", "Relaxed Clock: Uncorrelated Exponential" and "Relaxed Clock: Uncorrelated Lognormal". A strict clock assumes a global clock rate with no variation among lineages in a tree. However, biology is generally not that simple. Often the data will best fit a relaxed molecular clock model. Relaxed molecular clock models (There are other models - one under development that will be included in later versions of BEAUti will be the Random Local Molecular Clock model) assume independent rates on different branches, with one or two parameters that define the distribution of rates across branches. The relaxed molecular clock models in BEAST are called "uncorrelated" because there is no a priori correlation between a lineage's rate and that of its ancestor [Aldous DJ: Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. Statistical Science 2001, 16(1):23-34]. Note: To test MCMC chain performance in the first run of BEAST on a new data set, it is often a good idea to start with a relatively simple model. In the context of divergence dating this might mean running a strict molecular clock (with informative priors on either clock.rate, one or more tMRCAs, or treeModel.rootHeight). If BEAST can't produce an adequate

sample of the posterior under a simple model, then it is unlikely to perform well on more complicated substitution and molecular clock models (I say go ahead and try anyways).

- Uncorrelated Relaxed Lognormal or Exponential: (I used Lognormal) When using the relaxed molecular clock models, the rate for each branch is drawn from an underlying exponential or lognormal distribution. We recommend the use of the uncorrelated relaxed lognormal clock as this gives an indication of how clock-like your data is (measured by the ucld.stdev parameter). If the ucld.stdev parameter estimate is close to 0.0 then the data is quite clock-like. If the ucld.stdev has an estimated value much greater than 1.0 then your data exhibits very substantial rate heterogeneity among lineages. (If this parameteris 0 there is no variation in rates among branches. If this parameter is greater than 1 then the standard deviation in branch rates is greater than the mean rate.) This pattern will also be true for the coefficient of variation parameter (although this parameter isn't very sensitive).
- The strict molecular clock is the basic model for rates among branches supported by BEAST. Under this model the tree is calibrated by either:
  - 1. Specifying a substitution rate (this can be done either by fixing the mean substitution rate to a designated value or by using a prior on the clock.rate parameter in the "Priors" panel) or
  - 2. Calibrating the dates of one or more internal nodes (by specifying a prior on the tMCRA of a taxon subset or the treeModel.rootHeight). This allows the divergence dates of clades (defined either as a monophyletic grouping or as the tMRCA of a specified taxon subset) to be calculated based on the best fit of a single mutation rate across the whole tree.

**Priors:** Rather than copying a huge chunk of the manual, I'll refer you to page 9-14 of the manual. Unless you are interested in the population dynamics (and for this, you should have many samples for each of many different times), you should use the Coalescent: Bayesian Skyline as a tree prior. This assumes the fewest parameters on a prior that is essentially a nuisance unless you're interested in population dynamics. "Bayesian skyline plot (BSP) … calculates the effective breeding population size (Ne) through time (up to a constant related to the generation length in the time units of the analysis). However, the BSP should only be used if the data are strongly informative about population history, or when the demographic history is not the primary object of interest and a flexible coalescent tree prior with minimal assumptions is desirable. This coalescent-based tree prior only requires you to specify how many discrete changes in the population history are allowed. It will then estimate a demographic function that has the specified number of steps integrated over all possible times of the change-points and population sizes within each step to calculate a function of Ne through time [Drummond AJ, Rambaut A, Shapiro B, Pybus OG: Bayesian coalescent inference of past population dynamics from molecular sequences. Mol Biol Evol 2005, 22(5):1185-1192]. Two variants of the BSP are provided, the "Stepwise" model in which the population is constant between change-points and then jumps instantaneously and the "Linear" model in which the population grows or declines linearly between change-points." The number of groups appropriate is best determined by dividing your number of taxa by 5 (use only integers, no decimals); you may not have more groups than you have nodes, and you

must have at least 2 groups. For the first time running a certain set of conditions, it's appropriate to use the default parameter settings for the Bayesian Skyline prior. You may wish to enforce more limiting boundaries on certain parameters if you need to increase your ESSs in future runs of the same data, but you can also just leave them as default (that is making the fewest assumptions).

**Operators:** Operators act on the given parameters in the BEAST analysis, determining how the MCMC chain proposes new states to move to. Appropriate choice of weights and tuning parameter values will allow the MCMC chain to reach equilibrium/stationary phase faster and sample the target distribution more efficiently. Each parameter in the substitution model has one or more operators. A scale operator scales the parameter up or down by a random scale factor, with the tuning parameter deciding the range of scale factors to choose from. The random walk operator adds or subtracts a random amount (_) to or from the parameter. Again the tuning parameter (window size, w) is used to specify the range of values that _ can take _ Uniform(−w/2,w). The uniform operator simply proposes a new value uniformly within a given range. The tuning column gives the tuning setting to the operator. Some operators do not have a tuning setting so have a N/A. Changing the tuning setting will set how large a move that operator will make which will affect how often that change is accepted by the MCMC algorithm. The weight column specifies how often each operator is used to propose a new state in the MCMC chain. Some parameters have very little interaction with the rest of the model and as a result tend to be estimated very efficiently. An example of such a parameter is hky.kappa which though efficiently estimated, requires a complete recalculation of the likelihood of the data whenever it is changed. Giving these parameters lower weights can improve the computational efficiency of the run. The efficiency of the MCMC chain can often be improved by altering the weight of the operators that work on the treeModel. For example, if you are analyzing x sequences, a very rough rule of thumb is that you should set the weight of each of: upDownOperator; uniformOperator on internalNodeHeights; narrowExchangeOperator; and subtreeSlideOperator to x/2. You should also set the weight of the wilsonBaldingOperator and the wideExchangeOperator to min(1, x/10). These are rough guidelines and other weights may work better, but we have found these guidelines to work reasonably well.

- Auto-optimize: The "auto optimize" option will automatically adjust the tuning parameters of operators as the MCMC algorithm runs, to try to achieve maximum efficiency. We recommend that you choose this option even if you've already "optimized" your run using previous tuning suggestions. The criterion that our auto-optimization method uses is a target acceptance probability for each operator. The target acceptance probability is 0.25 for most operators. The idea is that proposals for new parameter values should be bold enough that they explore the parameter space rapidly, without being so large that the proposals are never accepted. By tuning an operator so that it is accepted 25% of the time we find that the moves are big enough to explore the space while still allowing regular changes to the MCMC state. At the end of the MCMC run, a report on the performance of the operators will be given in the BEAST console. This report includes the proportion of times that each operator was accepted, the final values of these tuning settings and whether they were at the right level for the analysis and suggestions for changes to these values. These operator tuning parameters can be changed in order to minimize the amount of time taken to reach optimum performance in subsequent runs. Note:

changing the tuning parameters of the operators will not change the results of the analysis - it will only affect the efficiency of the sampling of the posterior distribution. Better tuning parameters (and weights) will lead to faster convergence and better mixing of the MCMC chain, which means that the MCMC run can be run for fewer generations to achieve the same Effective Sample Size (ESS).

**MCMC:** The MCMC panel allows you to set the number of generations the MCMC algorithm will run for, how often the data is logged to file and to name the output files that BEAST will store the data in.

- Length of chain: This is the number of generations that the MCMC algorithm will run for. The length of chain depends on the size of the data set, the complexity of the model and the quality of the sample required. The aim of setting the chain length is to achieve a reasonable ESS, especially for the parameters of interest. A very very rough rule of thumb is that for x taxa/individuals you need a generation time/chain length proportional to $x^2$. Thus if, for 100 sequences, a 30,000,000 step chain gives good results then for 200 similar sequences we may need a chain of 120,000,000 steps. If you don't have enough generations to get reasonable sample sizes the first time, you can run again (for fewer, same amount, or more generations) using the same conditions and then combine the results using LogCombiner.
- Logging options: The "Echo state to screen" option determines how often a summary of the state is outputted to the BEAST console window (e.g., every 1,000 generations). This option is only important for people that have enough spare time to monitor BEAST's progress as it runs ;-). The "Log parameters every" option determines how often parameter values are written to the log file (e.g., every 100 generations). Dividing the number of generations in the MCMC chain by the value specified for "Log parameters every", will give you the sample size at the end of the BEAST run. Ideally, you should aim for a sample size of between 1000 and 10,000 logged parameter values. Logging more samples will simply produce very large output files which may be difficult to analyze in other programs. NOTE: it's simplest to use the LogCombiner option when your generations are logged at the same frequency; to make sure I can always combine my runs, I just log every 1000 generations for all my runs, and I just deal with the big files. Sometimes they're too big to fit on a USB stick, though.
- File names: The log file name, tree file name and substitution trees file name determine where the data that BEAST creates will be saved. The log file will contain the posterior sample of the parameter values specified in the BEAST XML file. The tree file will contain a posterior sample of trees (with branch lengths in chronological units) that can be viewed in TreeView or FigTree. The (subst) tree file will be a tree file with the branch lengths in units of substitutions. These will be saved in the same folder that the BEAST XML file is saved under (on UNIX/Linux systems these files will be saved in the current working directory).

**Generating BEAST file and saving BEAUti template:** Finally, once you are satisfied that you have specified everything you want in the BEAST XML file, click on the "Generate BEAST File" button in the bottom right hand corner of the BEAUti window. This will generate an XML file that can be saved in a specific folder. This is the file that will be used by BEAST to execute the MCMC analysis. You can save a separate BEAUti file by selecting the "Save" option from the "File" menu. It will also be an XML file but

will not be recognized by BEAST, and is only used so that you can re-load it in BEAUti and quickly make modifications to your analysis at a later date. It is recommended that you save the BEAUti files with the extension ".beauti" to distinguish them from the BEAST input files. NOTE: Personal experience has been that very complicated template files won't save (e.g., lots of taxon sets for dating). I think this is a bug, but it's annoying. If you use the manual, it's fairly easy to make changes to the .xml file itself. Just open it in something like Notepad or Simpletext, edit whatever you want to change (*e.g.,* tuning parameters, # of generations), and save the file as an .xml file.

Often, one data set will be the source for many, many BEAST runs. I've found it most efficient to number each of my runs and assign a folder for all the files associated with each run, including the .xml BEAST file, the .beauti template, and the tree files and the log file produced by BEAST. I also include a simple text file in which I make notes about the purpose of the run, changes I've made since the last run, and settings I choose (and why).

**Specifying a starting tree:**

If a starting tree is left unspecified, BEAST uses Bayesian methods to explore all the possible tree space permitted using the data. However, there are several reasons you might want to specify a starting tree:

1. It can help the analysis achieve convergence a lot more quickly
2. You are reasonably certain that some aspects of the tree topology are real and should be constrained
3. Occasionally, a taxon will not be securely tied to a particular lineage, and may flip between a couple different lineages (these taxa will typically have somewhat lower bootstrap values in phylogenies produced using other methods). When this happens, it could result in substantially different branch lengths, leading to real difficulties in achieving convergence. This problem might be signaled by low ESS values even though the number of generations is quite high, and/or bi-modal or multi-modal histograms for parameters such as posterior probability, likelihood, and rate estimators. My solution for this problem is to constrain the taxon to monophyly in a particular lineage, if there's a reason to believe that the taxon belongs there. If not, then it would be reasonable to constrain the taxa to monophyly in each of the problem lineages, then compare the results of runs (perhaps do a likelihood ratio test to determine which would models the data better).
    - it appears that your MCMC chain is jumping between two quite different possible explanations of your data. In one explanation the rate have quite extreme rate variation among branches, in the other the rate variation among branchs is much more clock like. Since your chain has this bimodality and its mixing very slowly, you could probably look at the trace of ucld.stdev and visually pick out different parts of your chain that are in the different

modes. Then you could extract these parts of the chain from the tree file and use tree annotator to summarize them separately. This will allow you to look at the trees and rates associated with each of the two modes. It may be that one of the modes makes no biological sense and can be excluded by changing your priors (for example one mode might have the root position in the wrong place -- in which case you can force your ingroup to be monophyletic to ensure this doesn't happen). If one of the modes doesn't make sense you could also exclude it by putting limits on the ucld.stdev parameter..."

When you define a starting tree, BEAST will still explore the tree space (within the constraints of any parameters you have defined). If you are certain of your phylogeny, you can prevent BEAST from exploring other trees by defining a starting tree and deliberately removing the tree operators.

- How can you keep this topology constant while estimating other parameters, e.g., node height? Remove all the operators that act on the <treeModel>. This will be the <narrowExchange>, <wideExchange>, <wilsonBalding> and <subtreeSlide> operators. Without these operators the actual topology of the tree won't change.

It's really important that your starting tree has been obtained using a program OTHER than BEAST, or it won't be considered an "independent" starting tree.

From manual: "Starting with a random starting tree can sometimes lead to difficulties because the randomly generated tree may not satisfy all of the constraints that have been placed on it in the form of priors on tree topology and divergence times. If there are no hard priors (i.e., uniform priors on divergence times or monophyly constraints), then a random starting tree can be generated using the coalescent tree prior. The coalescent starting tree block has a unique id. It uses the taxa and demographic model element that you have specified to construct a random starting tree. This element will be generated based on options in the "Model" panel in BEAUti."

---

This is a piece of an XML file that does not specify a user-defined starting tree (but does use a coalescent starting tree). The pertinent sections are grayed and boxed.

```
  </alignment>
  - <!--  The unique patterns for codon positions 1 & 2                    -->
- <mergePatterns id="patterns1+2">
  - <!--  The unique patterns for codon position 1                         -->
```

```
  - <!-- npatterns=229 -->
- <patterns id="patterns1" from="1" every="3">
  <alignment idref="alignment" />
    </patterns>
  - <!-- The unique patterns for codon position 2 -->
  - <!-- npatterns=157 -->
- <patterns id="patterns2" from="2" every="3">
  <alignment idref="alignment" />
    </patterns>
    </mergePatterns>
  - <!-- The unique patterns for codon position 3 -->
  - <!-- npatterns=782 -->
- <patterns id="patterns3" from="3" every="3">
  <alignment idref="alignment" />
    </patterns>
  - <!-- This is a simple constant population size coalescent model -->
  - <!-- that is used to generate an initial tree for the chain. -->
- <constantSize id="initialDemo" units="years">
- <populationSize>
  <parameter id="initialDemo.popSize" value="100.0" />
    </populationSize>
    </constantSize>
  - <!-- Generate a random starting tree under the coalescent process -->
- <coalescentTree id="startingTree">
  <taxa idref="taxa" />
  <constantSize idref="initialDemo" />
    </coalescentTree>
- <treeModel id="treeModel">
  <coalescentTree idref="startingTree" />
- <rootHeight>
  <parameter id="treeModel.rootHeight" />
    </rootHeight>
- <nodeHeights internalNodes="true">
  <parameter id="treeModel.internalNodeHeights" />
    </nodeHeights>
- <nodeHeights internalNodes="true" rootNode="true">
  <parameter id="treeModel.allInternalNodeHeights" />
    </nodeHeights>
    </treeModel>
```

This is a piece of an XML file that does not specify a user-defined starting tree, but does constrain some groups to be monophyletic (this is done in BEAUti – see above). The pertinent sections are grayed and boxed.

```
    </alignment>
    - <!-- The unique patterns for codon positions 1 & 2 -->
  - <mergePatterns id="patterns1+2">
    - <!-- The unique patterns for codon position 1 -->
    - <!-- npatterns=172 -->
  - <patterns id="patterns1" from="1" every="3">
    <alignment idref="alignment" />
      </patterns>
    - <!-- The unique patterns for codon position 2 -->
    - <!-- npatterns=123 -->
  - <patterns id="patterns2" from="2" every="3">
    <alignment idref="alignment" />
      </patterns>
      </mergePatterns>
    - <!-- The unique patterns for codon position 3 -->
    - <!-- npatterns=546 -->
  - <patterns id="patterns3" from="3" every="3">
    <alignment idref="alignment" />
      </patterns>
    - <!-- This is a simple constant population size coalescent model -->
    - <!-- that is used to generate an initial tree for the chain. -->
  - <constantSize id="initialDemo" units="years">
  - <populationSize>
    <parameter id="initialDemo.popSize" value="100.0" />
      </populationSize>
      </constantSize>
    - <!-- Generate a random starting tree under the coalescent process -->
  - <coalescentTree id="startingTree">
  - <constrainedTaxa>
    <taxa idref="taxa" />
  - <tmrca monophyletic="false">
    <taxa idref="all CHIKV" />
      </tmrca>
  - <tmrca monophyletic="false">
    <taxa idref="ECSA/Asian" />
      </tmrca>
  - <tmrca monophyletic="false">
    <taxa idref="Old ECSA" />
      </tmrca>
  - <tmrca monophyletic="false">
    <taxa idref="Current Outbreak" />
      </tmrca>
  - <tmrca monophyletic="false">
    <taxa idref="Asian" />
      </tmrca>
  - <tmrca monophyletic="false">
    <taxa idref="SouthEast Asian" />
      </tmrca>
```

```
- <tmrca monophyletic="false">
  <taxa idref="WAf" />
    </tmrca>
    </constrainedTaxa>
  <constantSize idref="initialDemo" />
    </coalescentTree>
- <treeModel id="treeModel">
  <coalescentTree idref="startingTree" />
- <rootHeight>
  <parameter id="treeModel.rootHeight" />
    </rootHeight>
- <nodeHeights internalNodes="true">
  <parameter id="treeModel.internalNodeHeights" />
    </nodeHeights>
- <nodeHeights internalNodes="true" rootNode="true">
  <parameter id="treeModel.allInternalNodeHeights" />
    </nodeHeights>
    </treeModel>
```

This is a piece of an XML file that DOES specify a starting tree; the pertinent sections are grayed and boxed. To insert a Newick tree:

- Replace whole "coalescentTree" block (from <coalescentTree id="**startingTree**"> to </coalescentTree> ) with a "newick" block ( from <newick id="**startingTree**" to </newick>)
- Change the tree model element to make sure that it references the correct starting tree. (In the "treeModel" block, immediately after the "treeModel id", replace <coalescentTree idref="**startingTree**" /> with <newick idref="**startingTree**" />)

The numbers after the each taxa name (*e.g.*, REU05nov:9.1E-5) are the branch lengths as annotated in a standard Newick tree. Immediately after opening the Newick block, you must define the units of the branch lengths (mine are in "substitutions," short for "substitutions per site"; these could also be years). The starting tree, in the Newick format, follows the unit definition.

```
  </alignment>
  - <!-- The unique patterns for codon positions 1 & 2 -->
- <mergePatterns id="patterns1+2">
  - <!-- The unique patterns for codon position 1 -->
  - <!-- npatterns=172 -->
- <patterns id="patterns1" from="1" every="3">
  <alignment idref="alignment" />
    </patterns>
  - <!-- The unique patterns for codon position 2 -->
  - <!-- npatterns=123 -->
- <patterns id="patterns2" from="2" every="3">
  <alignment idref="alignment" />
    </patterns>
    </mergePatterns>
  - <!-- The unique patterns for codon position 3 -->
  - <!-- npatterns=546 -->
- <patterns id="patterns3" from="3" every="3">
  <alignment idref="alignment" />
    </patterns>
  - <!-- This is a simple constant population size coalescent model -->
  - <!-- that is used to generate an initial tree for the chain. -->
- <constantSize id="initialDemo" units="years">
- <populationSize>
  <parameter id="initialDemo.popSize" value="100.0" />
    </populationSize>
    </constantSize>
  - <!-- my own starting tree -->
  <newick id="startingTree" units="substitutions">(((((((REU05nov:9.1E-5,(GERb07:9.17E-4,MAU06:3.42E-4,FRA06:8.0E-5,REU05dec:3.49E-4,REU05csf:4.03E-4):1.68E-4):1.7E-4,SEY05:4.23E-4,REU05may:9.1E-5):2.53E-4,(((USA06v11:6.78E-4,((((ITA07:0.001282,SLA07v20:6.65E-4):3.4E-4,INDI06rj:3.44E-4):1.58E-4,SLA07v17:8.28E-4,INDI06tn:3.3E-4):1.64E-4,INDI06hy:5.95E-4):1.6E-4,INDI06ap:3.29E-4):1.63E-4,INDI06mh:1.65E-4):2.48E-4,INDI06ka:7.82E-4):7.14E-4):0.01062,((UGA82v34:2.8E-4,INDI00:4.85E-4):0.002713,CAR80v1:0.003382):0.0038380000000000003):0.006947,DRC60v29:0.00905):7.35E-4,(((SAF76v13:2.48E-4,SAF76v24:1.62E-4):0.00555,TAN53v33:0.001232):2.46E-
```

```
4,SAF56v9:0.003077):0.006526):0.008951,(((((((((PHI85v18:1.77E-4,PHI85v7:7.9E-
5):0.001344,INDO83v8:5.76E-4):1.61E-4,INDO85v15:0.001686):6.75E-
4,(THAI88v6:0.003099,(THAI95v14:9.08E-4,THAI95v21:0.001411,THAI95v4:9.39E-
4):0.005791):7.39E-4):0.002216,THAI78v27:0.001772):4.69E-
4,THAI75v37:0.001939):0.003891,THAI62:0.0017439999999999999):0.001765,THAI58v30:8.82E-
4):0.001184,(INDI63v36:1.68E-4,(INDI63v22:7.0E-5,INDI63:0.001935):1.66E-4,(INDI73:5.07E-
4,INDI73v5:2.47E-4):0.00413):0.003438):0.032714):0.11052,((SEN66v2:2.75E-4,SEN66v16:4.34E-
4):0.0032129999999999997,(SEN83:0.005813,(NIG64v3:2.19E-4,NIG65v26:4.62E-
4):0.002565):0.002067):0.118907);</newick>
- <treeModel id="treeModel">
  <newick idref="startingTree" />
- <rootHeight>
  <parameter id="treeModel.rootHeight" />
    </rootHeight>
- <nodeHeights internalNodes="true">
  <parameter id="treeModel.internalNodeHeights" />
    </nodeHeights>
- <nodeHeights internalNodes="true" rootNode="true">
  <parameter id="treeModel.allInternalNodeHeights" />
    </nodeHeights>
    </treeModel>
```

---

## Running BEAST

If you don't care how fast BEAST runs (although fix #1 is pretty easy – see below), all you have to do is click on the BEAST.exe, open your .xml file, and let it chug away. You should know that it's pretty power-intensive, so don't plan to use any other power-sink programs (and everything else will probably run more slowly). You can monitor the means/distributions of your parameters of interest during the run using Tracer (see below). At the end of the run, there will be a report for the optimized tuning settings and suggestions for how to optimize your next run. Copy these and paste them into a text file for future reference.

There is no "pause" setting for BEAST; however, if BEAST either crashes or is accidentally closed during the run, all the data until that point is automatically saved (it's saved as it's logged). You do lose the optimization parameters that are reported at the end of the run, but that's a lot better than losing the whole run.

To run BEAST faster:

- Fix #1 (I think this is only applicable to Linux and Windows, not Mac): Go to the BEAST program file. Open the "lib" subfolder. Copy the "NucleotideLikelihoodCore.dll" file. Return to the BEAST program file (one folder up). Paste the file here (in the same folder as the beast.exe file). For reasons understood by Tzuni (the discoverer of this trick) and having something to do with Java and C++, this will make BEAST run a heck of a lot faster (30-50%, in my experience).
- Fix #2 (this is good for all three platforms, see below): If you run BEAST from the command line, it runs BLAZING fast (comparatively). You should make sure not to try and copy in the command prompt using Ctrl+C though, as that closes the window (and therefore stops the run). Also, the command prompt is case-sensitive.

- Windows XP: Go to the Start menu → "run" (it's on the right-hand side)
    1) Type: `C:\Program Files\Beast v1.4.7\bin\beast.cmd -working`
        - it's case-sensitive; also, note the spaces (after "Beast" and ".cmd")
        - If you don't have BEAST in your program files folder, click to open up the "bin" folder from whatever location, and substitute the text in the "address" line at the top of the folder for what I have written.
        - There's a space before the "-working" argument – this argument tells the program to save the files to the folder where the .xml file comes from. (In tech-speak, it changes the working directory to the one from which the .xml file comes.
    2) The command-prompt BEAST should pop up and ask you to input an .xml file.
- Windows Vista: Open the command prompt. (Go to the Start menu → all programs → accessories → command prompt.) (Note: Command Prompt is case-sensitive.)
    1) It should say "`C:\Users\(your username here)>`" – change directory to the bin folder in the BEAST program folder.
        - I have a copy of the program folder, renamed "BEAST program files," in a folder on my desktop labeled "Beast stuff," so I type:
            - `cd Desktop\Beast stuff\BEAST program files\bin`
        - If yours is in the Program Files folder, type:
            - `cd C:\Program Files\BEAST\bin`
    2) Now it should say "`C:\blahblah…\bin >`" Type:
        - `beast.cmd -working`
        - There's a space before the "-working" argument – this argument tells the program to save the files to the folder where the .xml file comes from. In tech-speak, it changes the working directory to the one from which the .xml file comes.
    3) The command-prompt BEAST should pop up and ask you to input an .xml file.
- Mac OS X:
    1) Download the Linux version of BEAST and unzip (leave the unzipped folder on the desktop – note that I renamed mine "linux Beast").
    2) Open the command prompt ("terminal"). (Open the hard drive → application → utilities → terminal)
    3) Change directory to the "lib" subfolder in the Linux beast folder. Type:
        - `cd Desktop/linux\ BEAST.v1.4.7/lib`
        - I think the slash in the wrong direction is to avoid some issue with either the word "linux" or the space.
    4) Then type:
        - `java -jar beast.jar -working`
        - Note those are dashes preceded by a space. The "-working" argument tells the program to save the files to the folder where the .xml file

comes from. In tech-speak, it changes the working directory to the one from which the .xml file comes.

5) The prompt should come up asking you for an .xml file.

---

Still to come:

Monitoring the run: Tracer

Are you done with the run? How to know

Interpreting the results

Fixing problems: Rerunning (LogCombiner)

Assessing the reliability of the estimate: run using an empty data set, run testing other models

Making pretty trees: FigTree