

GARLI Demonstration

Derrick Zwickl

zwickl@nescent.org

Demo key:

- **bold words** represent settings found in the configuration file
- “words in quotations” are generally words that will be typed by you, either at the command line or in a text file
- >lines beginning with a > designate individual steps for you to follow on the computer
- *italicized words* designate filenames or directories

Getting started

GARLI reads all of its settings from a configuration file. By default it looks for a file named *garli.conf*, but other configuration files can be specified as a command line argument after the executable (i.e., if the executable were named *garli*, type “*garli myconfig.conf*”). Note that many of the settings typically don’t need to be changed from their default values. We will only experiment with some of the interesting settings in this demo, but detailed discussions of all settings can be found in the GARLI manual.

The config file is divided into two parts : **[general]** and **[master]**. The **[general]** section specifies settings such as the dataset to be read, starting conditions, model settings and output files. To start running a new dataset, the only setting that *must* be specified is the **datafname**, which specifies the file containing the data matrix in phylip or nexus format. The **[master]** section specifies settings that control the functioning of the genetic algorithm itself, and typically the default settings can be used.

The dataset:

We will be using a small 29 taxon by 1218 nucleotide mammal dataset that was taken from a much larger dataset (44 taxa with 17K bases over 20 genes) presented by Murphy et al. (2001, Science 294(5550):2348-2351). The genes included here are RAG1 and RAG2 (Recombination Activating Gene). This is a very difficult dataset because the internal branches are quite short, and the terminals quite long. GARLI performs much less repeatable on this dataset than any other of similar size that I’ve tested on, so consider this a “worst case” dataset. There are definitely local topological optima. The trees inferred using this gene also don’t match our understanding of the relationships of these taxa in many places.

1. Start a basic nucleotide run

In general there isn’t much that needs to be changed in the config file to start a preliminary run. In this case I’ve made a number of changes from the defaults to make the example more instructive and a bit faster. You’ll still need to set a few of the minimal settings. Note that the configuration file specifies that the program perform two independent search replicates (**searchreps** = 2).

- >Go to the *basicNucleotide* directory
- >Open your *garli.conf* file in a text editor (try double clicking it and if possible tell your operating system to always open this type of file with an appropriate text editor)
- >Enter “murphy29.rag1rag2.nex” on the **datafname** line.
- >Set the output file prefix (**ofprefix**) to “run1”. This will tell the program to start all output files with “run1...”.

To make things interesting, we'll have different members of the class use different settings for the starting trees. You will be assigned to one of three groups, and should enter these settings accordingly:

Group1 (random starting tree)

- >Set **streefname** to “random”

Group 2 (very rough stepwise addition starting tree)

- >Set **streefname** to “stepwise”
- >Set **attachmentspertaxon** to “5”

Group 2 (thorough stepwise addition starting tree)

- >Set **streefname** to “stepwise”
- >Set **attachmentspertaxon** to “100”

Now we need to set the model of sequence evolution to use. I've already run MrModeltest on this dataset (you can peruse the output in the MrModelTest directory). The best fit model under the AIC criterion is SYM+I+G (6 rates of substitution, equal base frequencies, an estimated proportion of invariant sites, gamma-distributed rate heterogeneity). This is close to the default GTR+I+G model set in the configuration file, so we only need to change the equilibrium base frequency settings.

- >Change the **statefrequencies** setting to “equal”

>Save the file

>Start GARLI (note that this is how you will start the program henceforth in this demo):

- Windows: double click on the runGarli.bat file in the *basicNucleotide* directory
- OS X: double click on the runGarli.command file in the *basicNucleotide* directory
- If you are proficient, you can also start the program in the *basicNucleotide* directory from the command line

You will see a bunch of text scroll by, telling you various information about the dataset and the run that is starting. Most of this information is not very important, but if the program stops be sure to read it for any error messages. The output will also contain information about the progress of the initial optimization phase, and as the program runs it will continue to log information to the screen. This output contains the current generation number, current best lnL score, optimization precision and the generation at which the last change in topology or optimization precision occurred. All of the screen output is also written to a log file that in this case will be named *run1.screen.log*, so you can come back and look at it later.

2. Monitor an ongoing run

>Look in the directory that GARLI is running in, and note the files that have been created by the run.

First we will look at the *run1.log00.log* file, which logs the current best lnL, runtime and optimization precision over the course of the run. It is useful for plotting the lnL over time.

>Open *run1.log00.log* in a text editor (you should be able to double click it and choose to open it in an appropriate editor)

Next, we'll look at the file that logs all of the information that is output to the screen.

>Open *run1.screen.log* in a text editor

This file contains an exact copy of the screen output of the program. It can be useful when you go back later and what to know what you did. In particular, check the "Model Report" near the start to ensure that the program is using the correct model.

Now lets look at the current best topology. This is contained in a file called *run1.current.best.tre*. This file is updated every **saveevery** generations, so it is always easy to see the currently best tree during a search (don't use this as a stopping criterion and kill the run when you like the tree though!!)

>Open *run1.current.best.tre* in Figtree and examine the tree (you may be able to double click it an associate .tre files with Figtree)

>You may want to root the tree on the Opossum outgroup branch

3. Look at the final results a run

Hopefully at least one of the search replicates has finished by now. Let's take a look at how the topology and branch lengths changed over the entire run. The *run1.repl.treelog00.tre* file contains each successive better scoring topology encountered during the first search replicate. Note that this file can be interesting to look at, but in general will not be very useful to you.

>Open *run1.repl.treelog00.tre* in Figtree

NOTE: if you root the tree, Figtree unfortunately no longer allows you to step through the trees in the file, so don't reroot

>Click through all of the trees

Note how the tree slowly changes over the run.

We can also get other information from the *treelog* file.

>Open the *run1.rep1.treelog00.tre* file in a text editor.

You will see a normal Nexus treesblock. Each tree line includes comments containing the lnL of that individual, the type of topological mutation that created it (1=NNI, 4=SPR, 8=local SPR) and the model parameters of that individual. If you scroll through and look at the mutation types, you will probably notice that a mix of all three topological mutation types were creating better trees early on, but the local NNI mutations dominate at the end of the run. As a comment in each tree specification you will also see the model parameters that were tied to each tree during the run. They are specified with a simple code:

r = relative rate matrix

e = equilibrium frequencies

a = alpha shape parameter of gamma rate heterogeneity distribution

p = proportion of invariable sites

The information that you really want from the program are the best trees found in each search replicate and the globally best across all replicates. After each individual replicate finishes, the best trees from all of the replicates completed thus far are written to the *.best.all.tre*. When all replicates have finished, the best tree across all replicates is written to the *.best.tre* file.

We can evaluate and compare the results of our two search replicates in PAUP.

Once both replicates have finished:

>In the basicNucleotide directory, execute the *murph29.rag1rag2.nex* file in PAUP

>Execute the *run1.best.all.tre* file in PAUP (Yes, I mean execute NOT *gettrees*. This will load the tree and parameter values into PAUP because there is a *paup* block written to the file in addition to the *trees* block)

>In PAUP, type “*lscore*”.

The *paup* block contained in the tree file tells PAUP to score the tree without actually doing any optimization, using the exact parameter values and branch lengths estimated by GARLI. The lnL score you get for the better of the two replicates should be very near that output by GARLI. You will also see the score of the best tree from the other replicate, but it may not exactly match that output by GARLI because the model parameter values used were those estimated on the best tree.

Make note of the lnL scores with GARLI’s parameters and branch lengths here:

Replicate 1 _____

Replicate 2 _____

> In PAUP, type

“*lset userbr=no rmat=estimate base=equal rates=gamma shape=estimate pinv=estimate*” to tell PAUP to optimize the branch lengths and all parameters of the SYM+I+G model.

> In PAUP, type “*lscore*”. (This will take a few minutes.)

After a bit you will see the optimized lnL score and parameter values for each tree as estimated by PAUP, which will be very similar to what you saw when PAUP used GARLI's values. The values and score are sure to change a little upon full optimization because GARLI does not deterministically optimize the parameters while it runs. The scores reported by PAUP are the true maximized lnL of the trees that you should take note of.

Make note of the lnL scores as optimized by PAUP here:

Replicate 1 _____

Replicate 2 _____

We can also use PAUP to compare the two trees and see if (and by how much) they differ.

> In PAUP, type "treedist".

This will show what is termed the Symmetric Tree Distance or the Robinson-Foulds Distance between the trees. It is a measure of how similar the trees are, and is two times the number of groupings that appear in one tree and not the other. If the trees are identical, the distance will be zero. The maximal distance between two fully resolved trees is $2 * (\# \text{ sequences} - 3)$.

If the trees are different, we can calculate a consensus in PAUP and see exactly where they agree. Note that in general you should choose the best scoring tree as your ML estimate instead of a consensus.

>In PAUP, type "contree" to get a strict consensus of the trees, in which any conflicting portions of the topology are collapsed.

One final note is that you can obviously look at your final trees in Figtree or another tree viewer, but those do not allow for quantitative comparisons.

4. Constrain a search

If you looked carefully at any of the trees you've inferred (and are a good mammalogist), you may have noticed that the relationships are somewhat strange (and definitely wrong) in places. One relationship that this small dataset apparently resolves correctly is the sister relationship of Whale and Hippo. This relationship (often termed the "Whippo" hypothesis) was once controversial, but is now fairly well accepted. If we are particularly interested in this relationship we might want to know how strongly the data support it. One way of doing this would be simply by looking at the bootstrap support for it, but we might be interested in a more rigorous hypothesis test such as a simulation based parametric bootstrap test or a Shimodaira-Hasegawa test. We won't go into those here, but Goldman et al. (2000, Syst. Biol. 49:652-670) provide a detailed discussion of the various available tests for comparing topological hypotheses.

The first step in applying one of the topological hypothesis tests is to find the best topology that does NOT contain the Whippo relationship. This is done by applying a constrained topology search. In this case what we want a negative (also called converse) constraint that causes GARLI to search through tree space while avoiding any tree that places Whale and Hippo as sister.

GARLI allows constraints to be specified in a number of ways. We will do it by specifying the bipartition to be constrained using what I call the “dot-star” format. This is a simple way of specifying a particular taxonomic grouping that uses a string of periods and asterisks, with one character for each taxon. All of the taxa specified with periods are on one side of the branch being specified, and all specified with an asterisk are on the other. For example, to constrain a grouping of taxa 1 and 2 in a dataset of 8 taxa, the string would be “**.....” (this is equivalent to “..*****”). In this case the taxa we want to constrain are numbers 20 and 21, out of 29 total taxa.

>In the *constrainedNucleotide* directory, use a text editor to create a new (empty) text file named *whippoNegative.con*
>On the first line, type in the string specifying the (Whale, Hippo) grouping (i.e. 19 periods, 2 asterisks, 8 periods)

Note that GARLI will also take constraint trees, which is how PAUP reads constraints. Constraints can be either positive (MUST be in the inferred tree) or converse (also called negative, CANNOT be in the inferred tree). The constraint type is specified to GARLI by putting a + or – at the start of the constraint string.

>Add a - to the beginning of the** string that you just created, so that it looks like this:

-.....**.....

>Save the file

Now we need to tell GARLI to use the constraint. The *garli.conf* file in this directory has already been set up to be similar to the one we used during the unconstrained search earlier, so we only need to make minimal changes.

>In the *constrainedNuclotide* directory edit the *garli.conf* file and change **constraintfile** to “*whippoNegative.con*”

>Change the **ofprefix** setting to “constrainedRun1”

>Save the config file

>Start GARLI by double clicking the runGarli file

When the run finishes, note the difference in lnL between the best tree that you found earlier and the best constrained tree. This difference is a measure of how strongly the data support the presence of the (Whale, Hippo) group relative to its absence. Unfortunately we can’t simply do a likelihood ratio test here to test for the significance of the difference because we have no expectation for how this test statistic should be

distributed under the null. That is what the parametric bootstrap or a Shimodaira-Hasagawa test would tell us, but is well beyond the scope of this demo.

Enter the lnL difference between the best overall and best constrained trees here:

5. Do an analysis using an amino acid model

In addition to nucleotide-based analyses, we can analyze these protein-coding genes at the amino acid level. We might prefer this to nucleotide analyses under certain conditions, particularly when our sequences are quite dissimilar and silent substitutions at the third codon position are saturated. Unlike nucleotide models the amino acid models also implicitly take into account how well particular amino acids can substitute for one other in proteins. However, by ignoring silent substitutions amino acid models do throw out information that may be informative at some level, and that has the potential to worsen phylogenetic estimates.

One handy feature of GARLI is that it can do the translation from nucleotide to amino acid sequences internally. So, you can use the same alignment file as input for both the nucleotide and amino acid analyses. Note that this beta version (0.96b7) only implements the standard genetic code though.

As with nucleotide analyses, the first step is to choose a model. The program ProtTest is similar to ModelTest, and does a good job of this. One unfortunate fact is that ProtTest does require an amino acid alignment, so a nucleotide alignment will have to be translated first. The program is available at:

<http://darwin.uvigo.es/software/prottest.html>

In this case I already ran ProtTest on this dataset, and the model it chose is what is usually termed the Jones or JTT model (Jones, Taylor and Thornton, 1992). You can look at the ProtTest output in the ProtTest directory if you like.

Set up the config file:

- >In the basicAminoacid directory, open the *garli.conf* file
- >Change the **datatype** from “dna” to “codon-aminoacid” (codon-aminoacid tells GARLI to do the codon to amino acid translation internally. **datatype** = aminoacid should be used for actual amino acid alignments)
- >Change both the **ratematrix** and **statefrequencies** settings to “jones”
- >Change **invariantsites** to “none”
- >Change **ofprefix** to AARun1
- >Save the file
- >Start GARLI by double clicking the runGarli file

You will notice that the amino acid analysis runs significantly slower than the nucleotide one, so the config file is set to only do a single search replicate.

Once the run finishes (approx. 10-15 min.) you can use Figtree to visually compare the trees from the nucleotide and amino acid analyses, or use PAUP for more quantitative comparisons. You will notice that the two types of analyses give very different results. In this case the nucleotide results are much more reasonable. You might also notice that the amino acid data do not support the Whippo relationship.

6. Bootstrap analyses

It won't be practical to run bootstrap analyses during an in-class exercise, but you will find the files resulting from a GARLI bootstrap analysis of the Murphy29 dataset in the *bootstrapNucleotide* directory. When performing a bootstrap analysis, GARLI will generate a number of bootstrap resampled datasets and perform a full tree search on each. The single best tree resulting from each replicate is written to a file named `<ofprefix>.boot.tre`.

GARLI does not currently calculate the bootstrap proportions or the bootstrap consensus tree from the set of trees found over the bootstrap replicates. An external program must be used for that, and good options are PAUP, the CONSENSE program from the Phylip package and a program called Bootscore (it requires a Python installation, and is available at <http://sourceforge.net/projects/bootscore>).

Since we are already familiar with it, we will post-process our bootstrap results using PAUP:

>In the *bootstrapNucleotide* directory, execute the `murphy29.rag1rag2.nex` data in PAUP

>In PAUP, load the bootstrap trees with the command
“`gettrees file=nucboot.boot.tre`”

>In PAUP, make the majority rule consensus tree
“`contree /strict=no majrule=yes LE50=yes grpfreq=no`”

This will display the majority-rule bootstrap consensus tree, including branches appearing in less than 50% of the trees (LE50). You will notice that some parts of the tree are very poorly supported, while others have high support. It is somewhat comforting that the parts of the tree that we know are resolved incorrectly receive low support. This is precisely why phylogenetic estimates MUST be evaluated in light of some measure of confidence, be it bootstrap values or posterior probabilities.

In the *bootstrapAminoacid* directory you will find the results of an amino acid based bootstrap run. You can follow the same procedure as above to obtain a bootstrap consensus of those data. You will notice that the amino acid support is generally very low.

7. Use checkpointing

One useful feature of the program is the ability to write “checkpoint” files during a run. If the run is stopped before finishing, either intentionally (e.g., need to reboot system, someone else needs to use system) or unintentionally (e.g., system crash, careless coworker closes it), it may then be restarted from the checkpoint and resume approximately where it left off. Note that a run restarted this way will give EXACTLY the same result that it would if never terminated.

- >In the *checkpoint* directory, edit the configuration file and set **writecheckpoints** to 1
- >Start GARLI by double clicking the runGarli file
- >After the run progresses for a minute or two, stop it by pressing control-C, closing the window that it is running in or killing it in the task manager.
- >Edit the configuration file again and set **restart** to 1
- >Start GARLI again
- >Note that it restarts from about where it left off

A run can be stopped and restarted with checkpointing as many times as you like. If you stop and restart it again no further changes would need to be made to the config file.

8. Further exercises

If you have your own dataset of interest, now would be a good time to give it a shot in GARLI. You can use the config file in the *basicNucleotide* directory as a template.

You might also try doing a constrained amino acid search that forces Whale and Hippo to be sister, since they are not sister in the ML amino acid tree. You can use the same constraint file as before, simply change the – at the start of the constraint string to a + to denote that it is a positive constraint. Comparing the likelihood of the constrained and unconstrained amino acid trees will give a measure of the support of the amino acid data against the Whippo hypothesis.