Bio::Tools::Phylo::PAML HOWTO

Aaron Mackey, *University of Virginia* [http://www.virginia.edu]
<amackey@virginia.edu>

Jason Stajich, *Duke University* [http://www.duke.edu]
<jason-at-bioperl.org>

This document is copyright Aaron Mackey, Jason Stajich, 2002-2005. For reproduction other than personal use please contact us at amackey@virginia.edu or jason@bioperl.org.

2002-08-01

Revision History

Revision 0.1	2002-08-01	ajm
first draft		
Revision 0.2	2003-03-01	jes
Added pairwise Ka,Ks example code and running code		
Revision 0.3	2005-03-15	jes
Added branch-specific paramater parsing (NSsites and per-branch rates)		

paml is a package of C programs that implement *P*hylogenetic *A*nalyses using *M*aximum *L*ikelihood, written by Dr. Ziheng Yang, University College London. These programs implement a wide variety of models to explore the evolutionary relationships between sequences at either the protein, codon or raw DNA level. This document's aim is to explore and document how the BioPerl *paml* parser and result objects "work".

Table of Contents

1. Background

The *paml* package consists of many different executable programs, but the BioPerl Bio::Tools::Phylo::PAML object (hereafter referred to as simply the *paml* object) focuses on dealing with the output of the main analysis programs "baseml", "codeml" (sometimes called "aaml") and "codemlsites" (a batch version of "codeml"). All of these programs use maximum likelihood methods to fit a mathematical model of evolution to sequence data provided by the user. The main difference between these programs is the type of sequence on which they operate (baseml for raw DNA, codeml for DNA organized as codons, aaml for amino acids).

While the general maximum likelihood approach used by the *paml* programs is the same for all of them, the specific evolutionary models available for each sequence type vary greatly, as do the parameters specific to each model. The programs function in a handful of disparate modes, each requiring slight variations of inputs that can possibly include:

1. multiply-aligned sequences. representing 1 or more distinct genes [*paml* parameter *Mgene* = 1], in 1 or more distinct datasets [*paml ndata* > 1])

2. a user-provided tree topology (or multiple tree topologies to be evaluated and contrasted)

3. a set of instructions in a control file that specify the model (or models) to be used, various options to specify how to handle the sequence data (e.g. whether to dismiss columns with gaps or not [*cleandata* = 1]),

initial or fixed values for model parameters, and the filenames for other input data.

The output from *paml* is directed to multiple "targets": data is written to the user-specified primary output file (conventionally named with an .mlc extension), as well as various accessory files with fixed names (e.g. `2ML.t`, `2ML.dN`, `2ML.dS` for pairwise Maximum Likelihood calculations) that appear in the same directory that the output file is found.

The upshot of these comments is that one *paml* program "run" can potentially generate results for many genes, many datasets, many tree toplogies and many evolutionary models, spread across multiple output files. Currently, the *paml* programs deal with the various categories of multiple analyses in the following "top-down" order: datasets, genes, models, tree topologies. So how shall the BioPerl *paml* module treat these sources of multiple results?

2. Accessing *paml* results

The BioPerl *paml* result parser takes the view that a distinct "recordset" or single, top-level PAML::Result object represents a single *dataset*. Each PAML::Result object may therefore contain data from multiple genes, models, and/or tree topologies. To parse the output from a multiple-dataset *paml* run, the familiar "next_result" iterator common to other BioPerl modules is invoked.

Example 1. Iterating over results with next_result

```
use Bio::Tools::Phylo::PAML;

my $parser = new Bio::Tools::Phylo::PAML (-file => "./output.mlc",
                                          -dir  => "./",
                                          -ctlf => "./codeml.ctl");

while(my $result = $parser->next_result) {
    # do something with the results from this dataset ...
}
```

In this example, we've created a new top-level *paml* parser, specifying *paml*'s primary output file, the directory in which any other accessory files may be found, and the control file. We then trigger the parser to begin parsing the data, returning a new PAML::Result object for each dataset found in the output.

The PAML::Result object provides access to the wide variety of data found in the output files. The specific kinds of data available depends on which *paml* analysis program was run, and the mode and models employed. Generally, these include a recapitulation of the input sequences and their multiple alignment (which may differ slightly from the original input sequences due to the data "cleansing" *paml* performs), descriptive statistics of the input sequences (e.g. codon usage tables, nucleotide or amino acid composition), pairwise Nei and Gojobori (NG) calculation matrices (for codon models), fitted model parameter values (including branch-specific parameters associated with any provided tree topology), reconstructed ancestral sequences (again, associated with an accompanying tree topology), or statistical comparisons of multiple tree topologies.

3. Running PAML from within Bioperl

Bioperl also has facilities for running *paml* from within a Perl script. This allows you to compute Ka and Ks estimations from within an analysis pipeline. The following section will describe the process of getting data into Bioperl, running the alignment process, and setting up a *paml* process. This code is focusing on estimations of all the pairwise Ka and Ks values however it can be used to easily compute more sophisticated questions about variable rates, etc.

This code below is an excerpt from scripts/utilities/pairwise_kaks.PLS which will calculate all pairwise Ka,Ks values for a set of cDNA sequences stored in a file. It will first translate the cDNA into protein and align the protein sequences. This is a simple way to insure gaps only occur at codon boundaries and amino acid substitution rates are applied when calculating the MSA. The protein alignment is them projected back into cDNA co-

ordinates using a method called aa_to_dna_aln. Finally the cDNA alignment is provided to a *paml* executing module which sets up the running parameters and converts the alignment to the appropriate format.

```
use Bio::Tools::Run::Phylo::PAML::Codeml;
use Bio::Tools::Run::Alignment::Clustalw;

# for projecting alignments from protein to R/DNA space
use Bio::Align::Utilities qw(aa_to_dna_aln);

# for input of the sequence data
use Bio::SeqIO;
use Bio::AlignIO;

my $aln_factory = new Bio::Tools::Run::Alignment::Clustalw();

my $seqdata = 'cdna.fa';

my $seqio = new Bio::SeqIO(-file   => $seqdata,
                           -format => 'fasta');
my %seqs;
my @prots;
# process each sequence
while ( my $seq = $seqio->next_seq ) {
    $seqs{$seq->display_id} = $seq;
    # translate them into protein
    my $protein = $seq->translate();
    my $pseq = $protein->seq();
    if( $pseq =~ /\*/ &&
        $pseq !~ /\*$/ ) {
        warn("provided a cDNA sequence with a stop codon, PAML will choke!");
        exit(0);
    }
    # Tcoffee can't handle '*' even if it is trailing
    $pseq =~ s/\*//g;
    $protein->seq($pseq);
    push @prots, $protein;
}

if( @prots < 2 ) {
    warn("Need at least 2 cDNA sequences to proceed");
    exit(0);
}

open(OUT, ">align_output.txt") ||
      die("cannot open output $output for writing");
# Align the sequences with clustalw
my $aa_aln = $aln_factory->align(\@prots);
# project the protein alignment back to cDNA coordinates
my $dna_aln = &aa_to_dna_aln($aa_aln, \%seqs);

my @each = $dna_aln->each_seq();

my $kaks_factory = new Bio::Tools::Run::Phylo::PAML::Codeml
                  ( -params => { 'runmode' => -2,
                                 'seqtype' => 1,
                               }
                  );

# set the alignment object
$kaks_factory->alignment($dna_aln);

# run the KaKs analysis
my ($rc,$parser) = $kaks_factory->run();
my $result = $parser->next_result;
my $MLmatrix = $result->get_MLmatrix();

my @otus = $result->get_seqs();
# this gives us a mapping from the PAML order of sequences back to
```

```
# the input order (since names get truncated)
my @pos = map {
    my $c= 1;
    foreach my $s ( @each ) {
        last if( $s->display_id eq $_->display_id );
        $c++;
    }
    $c;
} @otus;

print OUT join("\t", qw(SEQ1 SEQ2 Ka Ks Ka/Ks PROT_PERCENTID CDNA_PERCENTID)), "\n";
for( my $i = 0; $i < (scalar @otus -1) ; $i++) {
    for( my $j = $i+1; $j < (scalar @otus); $j++ ) {
        my $sub_aa_aln  = $aa_aln->select_noncont($pos[$i],$pos[$j]);
        my $sub_dna_aln = $dna_aln->select_noncont($pos[$i],$pos[$j]);
        print OUT join("\t",
                       $otus[$i]->display_id,
                       $otus[$j]->display_id,$MLmatrix->[$i]->[$j]->{'dN'},
                       $MLmatrix->[$i]->[$j]->{'dS'},
                       $MLmatrix->[$i]->[$j]->{'omega'},
                       sprintf("%.2f",$sub_aa_aln->percentage_identity),
                       sprintf("%.2f",$sub_dna_aln->percentage_identity),
                       ), "\n";
    }
}
```

4. Parsing branch-specific rates and NSSites results

PAML allows for several models of molecular evolution. Given a tree topology one can test whether or not constraints of evolutionary rates on different parts of the topology better explain the observe data than a null model where an overall rate is assumed. To do this a tree topology must be provided and typically marked to specify which branches to test the alternative hypotheses of differing rates.

To get access to this branch specific data we store it in the `Bio::Tree::Tree` object which is parsed for each result. The nodes in the tree will contain additional tagged values for capturing the branch specific evolutionary rates. In cases where there are several different models of evolution tested (i.e. M0, M1, etc) we create a `Bio::Tools::Phylo::PAML::ModelResult` to store each of the model results separately. A separate Tree object will be associated with each one of these models.

Please note that the models underlting PAML 3.14 have changed some from PAML 3.13 and earlier. NSsites 1 and 2 mean slightly different things than they did in previous versions.

Accessing Tree data

First we'll just descibe how to access data for a topology for a single model or where NSsites=0. In this case we'll just want to get the tree(s) associated with a give result. In this code we loop through all the `Bio::Tree::Tree` associated with the `Bio::Tools::Phylo::PAML::Result`.

```
use Bio::Tools::Phylo::PAML;

my $outcodeml = shift(@ARGV);
my $paml_parser = new Bio::Tools::Phylo::PAML(-file => $outcodeml,
                                              -dir => "./");
if( my $result = $paml_parser->next_result() ) {
 while ( my $tree = $result->next_tree ) {
  for my $node ( $tree->get_nodes ) {
   my $id;
   # first we do some work to figure out what the ID should be.
   # for a leaf or tip node this is just the taxon label
```

```
   if( $node->is_Leaf() ) {
    $id = $node->id;
   } else {
   # for the internal nodes it is just the name of all the sub-nodes
   # put together, much like how Sanderson represents internal nodes
   # in r8s
    $id = "(".join(",", map { $_->id } grep { $_->is_Leaf }
                           $node->get_all_Descendents) .")";
   }
   if( ! $node->ancestor || ! $node->has_tag('t') ) {
     # skip when no values have been associated with this node
     # (like the root node)
     next;
   }
   printf "%s\tt=%.3f\tS=%.1f\tN=%.1f\tdN/dS=%.4f\tdN=%.4f\t".
          "dS=%.4f\tS*dS=%.1f\tN*dN=%.1f\n",
    $id,map { ($node->get_tag_values($_))[0] }
    qw(t S N dN/dS dN dS), 'S*dS', 'N*dN';
  }
 }
}
```

Accessing NSsites data

In cases where nssites=1 or nssites=2 is provided the data for the results is accessible through the `Bio::Tools::Phylo::PAML::ModelResult`. The function `get_NSSite_results` on the `Bio::Tool::Phylo::PAML::Result` object. In this way multiple model results can be folded into a single `PAML::Result` object. The code shown below is nearly identical to that in the previous example, there is just an additional loop to process the NSsite Result objects.

```
use Bio::Tools::Phylo::PAML;

my $outcodeml = shift(@ARGV);
my $paml_parser = new Bio::Tools::Phylo::PAML(-file => $outcodeml,
                                              -dir => "./");
if( my $result = $paml_parser->next_result() ) {
  for my $ns_result ( $result->get_NSSite_results ) {
    print "model ", $ns_result->model_num, " ",
          $ns_result->model_description, "\n";
  while ( my $tree = $ns_result->next_tree ) {
   for my $node ( $tree->get_nodes ) {
    my $id;
    # first we do some work to figure out what the ID should be.
    # for a leaf or tip node this is just the taxon label
    if( $node->is_Leaf() ) {
     $id = $node->id;
    } else {
    # for the internal nodes it is just the name of all the sub-nodes
    # put together, much like how Sanderson represents internal nodes
    # in r8s
     $id = "(".join(",", map { $_->id } grep { $_->is_Leaf }
                           $node->get_all_Descendents) .")";
    }
    if( ! $node->ancestor || ! $node->has_tag('t') ) {
     # skip when no values have been associated with this node
     # (like the root node)
     next;
    }
    printf "%s\tt=%.3f\tS=%.1f\tN=%.1f\tdN/dS=%.4f\tdN=%.4f\t".
          "dS=%.4f\tS*dS=%.1f\tN*dN=%.1f\n",
    $id,map { ($node->get_tag_values($_))[0] }
    qw(t S N dN/dS dN dS), 'S*dS', 'N*dN';
  }
 }
}
```

```
}
```