Implementing phylogenetic workflows for comparative genomics using BioPerl

Jason Stajich
University of California, Berkeley, USA
jason_stajich@berkeley.edu

Albert Vilella
European Bioinformatics Institute, Hinxton, UK
avilella@gmail.com

July 21, 2007
Introduction to phylogenetics workflows

**Introduction**

Species A  
Species B  
Species C  
Genomes

All vs All sequence similarity search  
Shared gene families  
Orthologous genes

Gene trees  
Lineage specific expansions/contractions  
Selection analysis  
Gene trees  
Selection analysis
Outline

• Research questions in Comparative Genomics
  – Automated Orthologous and Paralogous gene identification
  – Sequence evolution: adaptive, constrained, and neutral
  – Gene family evolution: lineage-specific changes

• Tools for comparative genomics
  – Sequence similarity & Gene family clustering
  – Multiple sequence alignment
  – Phylogenetics
  – Molecular evolution

• BioPerl for building Pipelines
  – Data conversion
  – Running external applications
  – Processing results
Comparative Genomics

- Comparisons to study evolutionary history of genomes
- Identify commonalities and differences between genomes
- Orthologous and unique genes among species
- Paralogous gene families
- Use similarity search and alignment tools to identify homologs
- Use phylogenetic approaches to reconstruct evolutionary history
Principles of molecular evolution

- Sequences that share significant similarity are likely homologous
- Homologous sequences often have the same function
- Identification of sequence differences and similarities can suggest regions with new or conserved functions
- Models of sequence evolution allow inference of rates of evolution
- Comparison of multiple genes and genomes can identify sequences evolving at significantly different rates
- Sequences or regions with different rates may be under different selective constraint and can suggest innovation or relaxation of pressure.
Detecting selection between species

- For aligned orthologous genes
- Using codon-based methods identify where rate of change is faster in Non-Synonymous ($K_A$) than in Synonymous ($K_S$).
Gene family evolution

- Changes in family content can be powerful for understanding species differences
  - 6% different between Humans and Chimps (Demuth et al, PLoS One 2006).
  - Hydrophobin expansion in basidiomycete mushrooms
  - *C. elegans* chemoreceptor family expansions (Chen et al, PNAS 2006)
  - Purine salvage enzyme HPRT1 family in vertebrates (Keebaugh et al, Genomics 2007)
Local expansion of chemoreceptor genes in *C. elegans*

*C. elegans* Chromosome V

Introduction to phylogenetics workflows

Tree of Hydrophobins in 3 fungi

Local duplications

umay UM05010
umay UM04433

ccin 10587
ccin 10586
ccin 05130
ccin 05145
ccin 00406
pchr 10481
pchr 10482
pchr 06735
pchr 08984
pchr 09319
pchr 09342
pchr 09062
pchr 09063
pchr 09064
pchr 09065
pchr 09066
pchr 09067
pchr 00495
pchr 08523
pchr 11384
pchr 11183
pchr 11134
pchr 00475
pchr 00499
ccin 13133
ccin 05197
ccin 05199
ccin 08203
ccin 08204
ccin 08205
ccin 08201
ccin 08202
ccin 08198
ccin 08199
ccin 08221
ccin 08222
ccin 08197
ccin 05197
ccin 05199
ccin 08657
ccin 08203
ccin 08204
ccin 08205
ccin 08201
ccin 08202
ccin 08198
ccin 08199
ccin 08221
ccin 08222
ccin 08197
ccin 05197
ccin 05199
ccin 08657
Introduction to phylogenetics workflows

Hydrophobin expansion driven by local duplications

*P. chrysosporium*

Curated Genes

*Named Genes CDS*

GLEAN models

- aug_GLEAN_04862
- Probability 1
- GLEAN.gz_06185
- Probability 1

Pfam domains

Hydrophobin

Fungal hydrophobin E-value: 6.8e-27

C. cinereus

GLEAN models

- GLEAN_08870
  - Probability 1
- GLEAN.gz_05060
  - Probability 0.999998

Pfam domains

Hydrophobin

Fungal hydrophobin e-value: 1.90001e-40
Definitions for sequence relationships

- **Homology** - Similar sequences that share a common ancestor.

- **Orthology** - Similar sequences that descended from a common ancestor through speciation events.

- **Paralogy** - Similar sequences which arose through a duplication event within a species lineage.

- Sequences are generally considered similar if they share at least 30% identity at the amino acid level.
Introduction to phylogenetics workflows

Species Tree and Gene Tree

Introduction to phylogenetics workflows

**Gene tree/Species tree reconciliation**

- **Parsimony**
  - For each node in the tree identify whether it arose via duplication or speciation minimizing the number of duplication events.

- **Maximum Likelihood and Bayesian frameworks**
  - Maximize likelihood of data given gene tree and species tree, inserting branches on gene tree to represent losses and gains.
Introduction to phylogenetics workflows

Orthology and Paralogy types

- Duplication node
- Speciation node

**Between species paralogy**: Mmus1:Hsap2

**Within species paralogy**:
- Hsap2:Hsap2' Homo sapiens
- Mmus2:Mmus2' Mus musculus
- Mmus3:Mmus3' Euarchontoglires

**Ortholog many2many**: Hsap2:Mmus2, Hsap2:Mmus2', Hsap2':Mmus2, Hsap2':Mmus2'

**Ortholog one2many**: Hsap3:Mmus3, Hsap3:Mmus3'

**Ortholog one2one**: Hsap1:Mmus1

**Reconciled gene tree**

**Multiple Sequence Alignment**

**Homology Inference**
Paralogous family creation through duplication

- Duplication may be substrate for novel function (Ohno)

- Mechanisms of duplications
  - Unequal crossing-over during recombination
  - Retrotransposition
  - Translocations of large regions

- Different mechanisms will create different patterns of duplication
  - Members of a family are Local and physically clustered
  - Family members are dispersed
  - Duplicated blocks of genes
Introduction to phylogenetics workflows

**Paralogous gene relationship and inference**

- **Duplication node**
- **Speciation node**

```
Mmus
  └── Rnor
    └── Hsap

Mmus:Rnor

ortholog_one2one
Hsap:Mmus

ortholog_one2one
Hsap:Rnor

apparent_ortholog_one2one
Mmus:Rnor

apparent_ortholog_one2one
Hsap:Rnor

Dubious Duplication
species_intersection_score=0
```

```
Software and Tools
Software, Tools, and Data sources

- Inferring Orthologous and Paralogous genes
- Aligning Sequences
- Phylogenetic inference and Building Trees
- Testing for Selection
- Evaluate gene family size changes
- Data sources
Orthology Determination

- Best reciprocal hits (or Best Bi-Directional Hits)
- Refinements of BRH
  - InParanoid
  - OrthoMCL
- Tree-based
  - SDI & RIO (Zmasek and Eddy) [Parsimony]
  - Softparsemap (Berglund et al) [Parsimony]
  - Notung (Vernot, Goldman, and Durand) [ML]
  - RAP (Dufayard, Duret, and Rechenmann) [ML]
  - primetv (Arvestad, Berglund, Lagergren, and Sennblad) [Bayesian]
  - NJTREE (Li et al) [Parsimony/soft constraining]
Best Reciprocal hits

Query: YALI0E03168g
BDEN_JAM81_05913  1.9e-23
BDEN_JAM81_01717  5.2e-20
BDEN_JAM81_02197  2.0e-16
BDEN_JAM81_09371  1.6e-12
BDEN_JAM81_07589  9.4e-11
BDEN_JAM81_03703  2.0e-09
BDEN_JAM81_07588  1.4e-07

Query: BDEN_JAM81_05913
Query: YALI0E03168g  1.2e-23
YALI0F08789g  2.7e-22
Gene Family Building

Using pairwise similarities from tools like BLAST and FASTA we can build gene family clusters.

• Single-Linkage - if $A \rightarrow B$ and $B \rightarrow C$, then a cluster would be formed of A,B,C.

• Jaccard clustering - used at TIGR and Celera. Essentially single-linkage but it has an additional ability to prune things that are too far away.

• MCL (TRIBE) - map sequence similarity into distances on a graph and manipulate the graph to find stable clusters of genes in a family.

• hcluster_sg (Treefam) - a hierarchical clustering software for sparse graphs. Hierarchical clustering under mean distance.
Software, Tools, & Data sources

Multiple Alignments

Given clusters of homologous sequences, one can examine their evolutionary history through construction of a multiple sequence alignment.

- ClustalW - progressive multiple aligner
- MUSCLE - progressive multiple aligner with log-expectation score
- T-Coffee - progressive multiple aligner with high accuracy
- ProbCons - probability consistent aligner
- MAFFT - Alignmener that uses Fast Fourier Transformation
Phylogenetic inference and Building Trees (1)

- Parsimony
  - PAUP* (aa or nt)
  - protpars, dnapars in PHYLIP (aa or nt)
  - LVB (nt)
  - TNT* (aa or nt)

- Distance based
  - protdist or dnadist + neighbor in PHYLIP (aa or dna)
  - BioNJ (aa or nt)
  - PAUP* (aa or nt)
  - NJTree (aa, codon, or nt)

* - Not freely available
Phylogenetic inference and Building Trees (2)

- **Maximum Likelihood**
  - PHYML (aa and nt)
  - PUZZLE (aa and nt)
  - ProtML (aa; very old)
  - dnaML (nt)
  - GARLI (nt)
  - RAxML (aa and nt)
  - PAUP* (nt)
  - P4 (nt)

- **Bayesian**
  - MrBayes (aa, codon, and nt)
  - PhyloBayes (aa and nt)
  - P4 (nt)
NJTree - Tree Merge

dN+dS
NJTree - Tree Merge

• ML-AA-WAG4 - WAG matrix amino acidic model - Maximum Likelihood

• ML-NT-HKY4 - Hasegawa-Kishino-Yano nucleotidic model - Maximum Likelihood

• NJ-NT-dN - non-syn substitutions - neighbor-joining with bootstrap

• NJ-NT-dS - synonymous substitutions - neighbor-joining with bootstrap
ML approaches to testing for selection

- Start with codon alignment (gaps are multiple of 3). Best to start with protein alignment then map.

- No stop codons!

- Most powerful with several sequences that have appropriate divergence (Anisimova et al, MBE 2002)

- **Not** designed for recombining populations/popgen data

- Tools
  - PAML
  - HyPhy
Gene family size changes

- **Computational Analysis of gene Family Evolution (CAFE)**

- [http://www.bio.indiana.edu/~hahnlab/Software.html](http://www.bio.indiana.edu/~hahnlab/Software.html)
  - Use gene number count in each family to identify significant lineage-specific contractions or expansions on a phylogeny
  - Estimate a duplication-death rate ($\lambda$)
  - Probabilistic graphical model for estimating rates and ancestral states
  - New models allow for hypothesis testing of multiple $\lambda$ rates
Data sources and types

• Data sources
  – Pre-analyzed EnsEMBL gene families
  – Genome sequences from GenBank or Genome Projects

• Data types
  – Need CoDing Sequences (CDS) not cDNA for selection analyses
  – For Ensembl this can be obtained through BioMart
  – Or get all gene annotations in GFF and derive CDS from annotated CDS exons.
  – Alternative splicing must be considered
  – For GenBank/EMBL files parse and retrieve CDS from annotated genes
Ensembl Compara

1. Load Genes and Longest Translation from all species in Ensembl
2. Build a graph of gene relations based on BRH and BSH
3. For each cluster, build a multiple alignment (MUSCLE) based on the protein sequences
4. Interence of ortholog and paralogs (OrthoTree)
5. WU Blastp + SmithWaterman longest translation of every Gene against every other in a genome-wise manner
6. Extract the connected components
7. From each alignment, build a reconciled gene tree with internal duplication nodes wrt a species tree (NJTREE_PHYML)
Gene trees in Ensembl Compara
Gene trees in Ensembl Compara

- Bio::EnsEMBL::Compara::ProteinTree
  - Tree topology + labeled duplication nodes

- Bio::EnsEMBL::Compara::NCBITaxon
  - Species trees with NCBI Taxonomy labels

- Bio::EnsEMBL::Compara::Homology
  - Homology description for each pair of genes
Querying Ensembl Compara database

```perl
#!/usr/bin/perl -w
use strict;
use Bio::EnsEMBL::Registry;

Bio::EnsEMBL::Registry->load_registry_from_db
  (-host => "ensembldb.ensembl.org",
   -user => "anonymous",
   -db_version => '44',
   -verbose => '0');

my $human_gene_adaptor = Bio::EnsEMBL::Registry->
  get_adaptor
    ("Homo sapiens", "core", "Gene");
my $member_adaptor = Bio::EnsEMBL::Registry->get_adaptor
  ("Compara", "compara", "Member");
```
my $homology_adaptor = Bio::EnsEMBL::Registry->get_adaptor (
    "Compara", "compara", "Homology");
my $genes = $human_gene_adaptor->fetch_all_by_external_name (
    'CTDP1');

foreach my $gene (@$genes) {
    my $member = $member_adaptor->fetch_by_source_stable_id
        ("ENSEMBLGENE",$gene->stable_id);
    my $all_homologies = $homology_adaptor->fetch_by_Member($member);

    foreach my $this_homology (@$all_homologies) {
        my $description = $this_homology->description;
        next unless ($description =~ /one2one/);
        my $all_member_attributes = $this_homology->get_all_Member_Attribute();
        my $first_found = 0;
foreach my $ma (@$all_member_attributes) {
    my ($mb, $attr) = @$ma;
    my $label = $mb->display_label || $mb->stable_id;
    print $mb->genome_db->short_name, ",", $label, "\t";
}
print "\n";
}
Introduction to BioPerl
Introduction to BioPerl

What is BioPerl?

• Perl modules toolkit for parsing data and results from Bioinformatics application

• International open-source project striving to reduce barrier to automating bioinformatics approaches

• Perl scripts and modules for running bioinformatics applications

• Interface to RDBMS storing sequence and feature data
Glue for Applications

Intro to BioPerl

- **Sequence database**
- **Genes in family**
- **Protein multi-sequence file**
- **Gene Tree**
- **Positively Selected genes**
- **Codon Alignment**
- **Multiple alignment**
- **Pruned alignment**

**Tools and Methods**
- Bio::DB::Fasta
- Bio::SeqIO
- Bio::AlignIO
- Bio::SimpleAlign
- PAML
- HyPhy
- PHYLIP
- PAUP
- PHYML
- MrBayes
- RAxML
- MUSCLE
- ClustalW
- T-Coffee
- Bio::Align::Utilities
Quick example: Sequence parsing

- Parse a database of sequences in FASTA format.

- For each sequence:
  - Print out the sequence ID.
  - Print out the first 20 residues of the sequence.
  - Print the length of the sequence.

- Print the total number of sequences
Quick example: Sequence file

FASTA database file with 1 sequence. "inputfile.fa"

>UM05311
mqlcvsnklr vsliwaccla lmglgapvsv efssslavfv drledasvpv lprqyhstwd
fssnkvhhlw tsflfkdgkn fnpklfylgr dplkldthla vahwypntfl lnrvrpigaq
tvnleyrhgl lamklashee pefvgilwik dkrnakkara adrakrsgkh reerresqgr
sksdppwdyl ievrcsldpl sitslstilv yihrictllr
Quick example: Sequence parsing code

```perl
use Bio::SeqIO;
my $input = Bio::SeqIO->new(-format => 'fasta',
                          -file => 'inputfile.fa');

my $seqcount = 0;
while( my $seq = $input->next_seq ) {
    print $seq->display_id, " is sequence ID.\n",
    substr($seq->seq,0,20),": 1st 20 residues of sequence.\n",
    $seq->subseq(1,20),  ": 1st 20 residues of sequence.\n",
    "It has ", $seq->length, " residues.\n",
    $seqcount++;
}
print "==\n",$seqcount, " total sequence(s) seen.\n";
```
Quick example: Sequence parsing result

UM05311 is sequence ID.
mqlcvsnklrvsliwaccla: 1st 20 residues of sequence.
mqlcvsnklrvsliwaccla: 1st 20 residues of sequence.
It has 220 residues.
==
1 total sequence(s) seen.
# Objects for Bioinformatics data

## Intro to BioPerl

<table>
<thead>
<tr>
<th>Data type</th>
<th>Example formats</th>
<th>BioPerl object class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence data</td>
<td>FASTA, GenBank, Swissprot</td>
<td>Bio::Seq</td>
</tr>
<tr>
<td>Sequence parser</td>
<td>BLAST, FASTA, HMMER</td>
<td>Bio::SeqIO</td>
</tr>
<tr>
<td>Similarity search</td>
<td>CLUSTAL, PHYLIP</td>
<td>Bio::SearchIO</td>
</tr>
<tr>
<td>Multiple alignment</td>
<td>NEWICK, NEXUS, NHX</td>
<td>Bio::AlignIO</td>
</tr>
<tr>
<td>Phylogenetic trees</td>
<td>GFF</td>
<td>Bio::TreeIO</td>
</tr>
<tr>
<td>Genomic features</td>
<td>GenBank, EMBL</td>
<td>Bio::FeatureIO</td>
</tr>
<tr>
<td>Sequence databases</td>
<td>PHYLIP protdist</td>
<td>Bio::DB::GenBank</td>
</tr>
<tr>
<td>Distance Matrix</td>
<td>BLAST, EMBOSS</td>
<td>Bio::Matrix::IO</td>
</tr>
<tr>
<td>Application wrappers</td>
<td>PDB</td>
<td>Bio::Tools::Run</td>
</tr>
<tr>
<td>Structures</td>
<td></td>
<td>Bio::Structure::IO</td>
</tr>
</tbody>
</table>
## Parsers for Bioinformatics application results

<table>
<thead>
<tr>
<th>Data type</th>
<th>BioPerl object class</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAML</td>
<td>Bio::Tools::Phylo::PAML</td>
</tr>
<tr>
<td>Genscan, Glimmer FgenesH</td>
<td>Bio::Tools::{Genscan,Glimmer,Fgenesh}</td>
</tr>
<tr>
<td>Genewise</td>
<td>Bio::Tools::Genewise</td>
</tr>
<tr>
<td>Primer3</td>
<td>Bio::Tools::Primer3</td>
</tr>
<tr>
<td>TmHMM</td>
<td>Bio::Tools::TmHMM</td>
</tr>
<tr>
<td>tRNAscanSE</td>
<td>Bio::Tools::tRNAscanSE</td>
</tr>
<tr>
<td>SignalP</td>
<td>Bio::Tools::SignalP</td>
</tr>
<tr>
<td>Sigcleave</td>
<td>Bio::Tools::Sigcleave</td>
</tr>
<tr>
<td>Sim4, Spidey</td>
<td>Bio::Tools::{Sim4,Spidey}::Results</td>
</tr>
</tbody>
</table>
Object-Oriented Perl and BioPerl

- Objects are modules
- Perl Object-Oriented (OO) can be a little confusing
- \texttt{new} signifies creation of a new object
- Need to use \texttt{Module} to use a particular Module
- Factories with plug-ins for the different parser systems
- Stream-based so that data in files need not assume a single entry per file.
Sequences & Features

• Features have locations on sequence

• Locations need not be contiguous - Exon locations on a genomic locus

• Features can have additional data associated (score, reading frame, etc)

• Sequences are Feature containers
• BioPerl SearchIO objects are currently optimized for getting all data from report. Working to make them more efficient, but aspects of object creation in Perl can make Bio::SearchIO a bottleneck.

• Tips for speed
  – WU-BLAST has many more options available for tweaking Sn and Sp.
  – Only generate what you need - if you don’t need alignments, consider the tabular output (NCBI -m 9; WUBLAST -mformat 3)
  – Only parse what you need - BioPerl has filters built in to allow you to only get back summary Hit objects, no need to build HSP alignments if they aren’t needed
BLAST parsing has three components

- **Results** - Bio::Search::Result
  - Query name, description, length
  - Search statistics and parameters

- **Hit** - Bio::Search::Hit
  - Hit id, description, length
  - Significance, E-value, bit score

- **HSP (Alignment)** - Bio::Search::HSP
  - Alignment start and end in query and subject coordinate
  - Alignment length, score, E-value
  - Sequence alignment, query, subject, and homology
use Bio::SearchIO;
my $in = Bio::SearchIO->new(-format => 'blast',
                           -file => 'result.bls');

while( my $result = $in->next_result ) {
    print $result->query_name, ' ', $result->
             query_description, "\n"
    print $result->database_name, ' ', $result->
             database_entries,
    " sequences and ", $result->database_letters, " residues\n"
    my $kappa = $result->get_statistic('kappa');
    print "kappa is $kappa\n";
}
use Bio::SearchIO;
my $in = Bio::SearchIO->new(-format => 'blast',
- file => 'result.bls');
while ( my $result = $in->next_result ) {
  while ( my $hit = $result->next_hit ) {
    print "Hit name is ", $hit->name, " ", $hit->
description, " ",
$hit->length, " ", $hit->significance, " ", $hit->score
, "\n";
  }
}

Intro to BioPerl

BLAST parsing sample code: HSP

```perl
use Bio::SearchIO;
my $in = Bio::SearchIO->new(-format => 'blast',
                          -file => 'result.bls');
while( my $result = $in->next_result ) {
    while( my $hit = $result->next_hit ) {
        while( my $hsp = $hit->next_hsp ) {
            printf "Q start..end=%d..%d; H start..end=%d..%d\n",
               $hsp->query->start, $hsp->query->end,
               $hsp->hit->start, $hsp->hit->end;
            printf "percent ID %d%%, %d%% query aligned\n",
               $hsp->percent_identity,
               100 * ($hsp->query->length / $result->query_length);
        }
    }
}
```
Exchanging search algorithms

• Because we’ve generalize the parsing, BLAST can be swapped with FASTA or SSEARCH results

• -format => ’fasta’

• Same general code will work, although sometimes additional methods (like SW score) are available.
Sequence extraction from data files

```
#!/usr/bin/perl -w
use strict;
use Bio::DB::Fasta;

my $fastafilename = 'sequences.fa';
my $db = Bio::DB::Fasta->new($fastafilename);

# simple access (no BioPerl objects created)
my $seq = $db->seq('AY007676', 1200 => 1301);
my $reverseseq = $db->seq('AY007676', 1301 => 1200);
print "forward: $seq\nreverse: $seq\n";
my @ids = $db->ids;
print 'The ids are ', join(',', @ids), "\n";
```
use Bio::Tools::GFF;
my $fio = Bio::Tools::GFF->new(-fh => "/*DATA
-gff_version => 3);
while( my $feature = $fio->next_feature ) {
    printf "%s:%d..%d %s
", $feature->seq_id,
$feature->strand > 0 ? ($feature->start, $feature->end) :
($feature->end, $feature->start), $feature->get_tag_values
('ID');
}

__DATA__
cimm_2.1 BI gene 3061047 3062290 . + . ID=CIMG_01174;Name=CIMG_01174.2
cimm_2.1 BI mRNA 3061047 3062290 . + . ID=CIMT_01174;Parent=CIMG_01174
cimm_2.1 BI cds 3061047 3061106 . + 0 ID=cimm_cds00001;Parent=CIMT_01174
cimm_2.1 BI cds 3061256 3062290 . + 0 ID=cimm_cds00002;Parent=CIMT_01174
Feature data from database

Not shown: Load GFF into a Bio::DB::GFF database with available bulk_load_gff script first.

```perl
use Bio::DB::GFF;

# Open the DB::GFF database
my $db = Bio::DB::GFF->new(-adaptor => 'dbi:mysql',
                           -dsn => 'dbi:mysql:elegans');

# fetch a 1 Mb segment of seq starting at landmark "ZK909"
my $segment = $db->segment('ZK909', 1 => 1000000);

# pull out all transcript features
my @transcripts = $segment->features('transcript');

# for each transcript, total the length of the introns
my %totals;

for my $t (@transcripts) {
    my @introns = $t->Intron;
    $totals{$t->name} += $_->length foreach @introns;
}
```
use Bio::DB::GenBank;
use Bio::DB::Query::GenBank;
my $query = "Arabidopsis[ORGN] AND topoisomerase[TITL] and 0:3000[SLEN]";
my $query_obj = Bio::DB::Query::GenBank->new
  (-db => 'nucleotide',
   -query => $query);
my $gb_obj = Bio::DB::GenBank->new;
my $stream_obj = $gb_obj->get_Stream_by_query($query_obj);
while ($seq_obj = $stream_obj->next_seq) {
  # do something with the sequence object
  print $seq_obj->display_id, "\t", $seq_obj->length, "\n";
}
use Bio::AlignIO;
my $input = Bio::AlignIO->new(-format => 'clustalw',
    -file => 'inputfile.aln');
my $out = Bio::AlignIO->new(-format => 'phylip',
    -file => 'output.phy');
while( my $aln = $input->next_aln ) {
    print $aln->no_sequences, " sequences are ",
        $aln->percentage_identity, "% identical\n";
    # a consensus string for the alignment
    my $consensus = $aln->consensus_string(50);
    # a consensus string representing which columns have gaps
    my $gap_line = $aln->gap_line;
    my $slice = $aln->slice(20,30);
    $out->write_aln($slice);
}
Multiple Sequence Alignments

- Reading and writing alignment formats
- Processing alignment to find gapped or poorly aligned columns
- Retrieve a slice of the alignment
- Remove columns from an alignment
- Concatenate alignments
- Calculate summary statistics like percent identity
- Map characters (replace '-' with '.')
- Compute a consensus sequence with a specified threshold of identity
- Compute a compact string (CIGAR) to represent the alignment in a database or GFF file
use Bio::AlignIO;
use Bio::SeqIO;
use Bio::Align::Utilities qw(aa_to_dna_aln);
my $input = Bio::AlignIO->new(-format => 'clustalw',
    -file => 'pep.aln');
my $out = Bio::AlignIO->new(-format => 'clustalw',
    -file => '>cds.aln');
my $aa_aln = $input->next_aln;
my $cds = Bio::SeqIO->new(-format => 'file',
    -file => 'cds.fa');
my %cds;
while( my $seq = $cds->next_aln ) {
    $cds{$seq->display_id} = $seq; }
my $cds_aln = &aa_to_dna_aln($aa_aln,%cds);
$out->write_aln($cds_aln);
Trees

- Trees are Nodes and Edges

- Nodes have pointers to parents (only 1) and children (0..N)

- Trees can be un-rooted or rooted

- Internal IDs **CAN** be bootstrap values, but the data formats do not dictate this. One must **KNOW** what information is encoded in internal node labels, the bootstrap() data will not be filled in automatically.
use Bio::TreeIO;
my $in = Bio::TreeIO->new(
  -format => 'newick',
  -file   => 'trees.tre');

while( my $tree = $in->next_tree ) {
  for my $node ( grep { ! $node->is_Leaf } $tree->get_nodes ) {
    next if ! $node->ancestor; # ignore the root node
    print "Node: ", $node->id, " length: ", $node->branch_length, " ";
    for my $child ( $node->get_Descendents ) {
      print "child: ", $child->id, " ", $child->branch_length, " ";
    }
    print "\n";
  }
}
Manipulating and Querying trees

- Manipulations
  - Re-root
  - Delete a single node or edge
  - Splice (remove) a set of nodes from the tree
  - Merge a lineage of nodes
  - Force a tree to be binary

- Calculations
  - Least Common Ancestor for a pair of nodes
  - Test if a set of nodes is monophyletic
  - Find the path path from a node to the tip or to the root
  - Search for a particular node by ID or other pattern
Running applications within BioPerl

Handles writing sequences, alignments, trees to a file in correct format, temporary file creation.

- Running BLAST
- Running EMBOSS tools
- Running PHYLIP
- Running PAML
Running BLAST Locally

use Bio::Tools::Run::StandAloneBlast;
use Bio::SeqIO;
# Local-blast "factory object" creation and blast
# parameter initialization:
my @params = (-database => 'swissprot',-outfile => 'blast1.out');
my $f = Bio::Tools::Run::StandAloneBlast->new(@params);
# Blast a sequence against a database:
my $str = Bio::SeqIO->new(-file=>'t/amino.fa',
    -format => 'fasta');
my $input = $str->next_seq();
# $input can also be a seq file in fasta format or an
arrayref of sequences
my $blast_report = $f->blastall($input);
Running BLAST Remotely at NCBI

use Bio::Tools::Run::RemoteBlast;
my $remote_blastxml = Bio::Tools::Run::RemoteBlast->new(
    '-prog' => 'blastp',
    '-data' => 'swissprot',
    '-readmethod' => 'xml', # tells the parser to use blastxml format for parsing
    '-expect' => '1e-5',
);

$remote_blastxml->retrieve_parameter('FORMAT_TYPE', 'XML'); # tells NCBI to send XML back

# change a query parameter
$remote_blastxml->submit_parameter('ENTREZ_QUERY', 'Mytilus californianus [ORGN]');
Running EMBOSS: water

```perl
use Bio::Factory::EMBOSS;
my $f = Bio::Factory::EMBOSS -> new(); # init factory
my $water = $f -> program( 'water' ); # get application
my $seq_to_test; # this would have a seq here
my @seqs_to_check; # list of seqs to compare
my $wateroutfile = 'out.water';
$water -> run({-sequencea => $seq_to_test,
    -seqall => @seqs_to_check,
    -aformat => 'msf',
    -gapopen => '10.0', -gapextend => '0.5',
    -outfile => $wateroutfile});
my $alnin = new Bio::AlignIO(-format => 'msf',
    -file => $wateroutfile);
my $aln = $alnin -> next_aln;
printf "%.2f\n", $aln -> overall_percentage_identity;
```
Running PHYLIP: Build parsimony tree

```perl
use Bio::Tools::Run::Phylo::Phylip::ProtPars;
use Bio::AlignIO;
my $alnio = Bio::AlignIO->new(-format => 'clustalw',
                            -file => 'cysprot.aln');
my $aln = $alnio->next_aln;

# Create the Tree
# using a threshold value of 30
my $tree_factory = Bio::Tools::Run::Phylo::Phylip::ProtPars->new
    (threshold => 30,
     jumble => "17,10",
     outgroup => 2);  # based on order of aln
my $tree = $tree_factory->run($aln);
```
Running PHYLIP: NJ Tree and Bootstrapping

use Bio::Tools::Run::Phylo::Phylib::Consense;
use Bio::Tools::Run::Phylo::Phylib::SeqBoot;
use Bio::Tools::Run::Phylo::Phylib::ProtDist;
use Bio::Tools::Run::Phylo::Phylib::Neighbor;

# next use seqboot to generate multiple alignments
my @params = ('datatype' => 'SEQUENCE', 'replicates' => 100);
my $seqboot_factory = Bio::Tools::Run::Phylo::Phylib::SeqBoot->new(@params);
my $aln_ref = $seqboot_factory->run($aln);

# next build distance matrices and construct trees
my $pd_factory = Bio::Tools::Run::Phylo::Phylib::ProtDist->new();
my $ne_factory = Bio::Tools::Run::Phylo::Phylib::Neighbor
foreach my $a (@{$aln_ref}){
    my $mat = $pd_factory->create_distance_matrix($a);
    push @tree, $ne_factory->create_tree($mat);
}

# now use consense to get a final tree
my $con_factory = Bio::Tools::Run::Phylo::Phylip::Consense->new();
$con_factory->outgroup('HUMAN');
my $tree = $con_factory->run($@tree);
PHYLIP: Long sequence names

- PHYLIP is hard coded to only handle sequence identifiers of length 10.
- Longer names require recompiling PHYLIP code
- Can also use code in BioPerl to safely handle long names
- Operates on Alignment
- Still have to restore names at the end of a run
use Bio::Tools::Run::Phylo::Phylip::SeqBoot;
use Bio::Tools::Run::Phylo::Phylip::ProtPars;
my ($aln_safe,$ref_name)=$aln->set_displayname_safe();
my $sb = Bio::Tools::Run::Phylo::Phylip::SeqBoot->new (@params);
my $pars = Bio::Tools::Run::Phylo::Phylip::ProtPars->new ( threshold => 30, jumble => "17,10");
my $tree = $tree_factory->run($aln_safe);
# Restore original sequence names on tree, after PHYLIP runs:
my @nodes = $tree->get_nodes();
foreach my $nd (@nodes){
    $nd->id($ref_name->{($nd->id_output)}) if $nd->is_Lef;
Running PAML: Pairwise dN and dS

```
use Bio::Tools::Run::PAML::Codeml;
use Bio::AlignIO;
#
# get a codon alignment from a file
my $alnio = Bio::AlignIO->new
    (-format => 'phylip', -file => shift @ARGV);
my $codon_aln = $alnio->next_aln;
my $kaks_f = Bio::Tools::Run::Phylo::PAML::Codeml->new
    ( -params => { 'runmode' => -2, 'seqtype' => 1, } );
$kaks_f->alignment($codon_aln);
my ($rc,$parser) = $kaks_f->run;
my $result = $parser->next_result;
my $MLmatrix = $result->get_MLmatrix();
my $pair1 = $MLmatrix->[0]->[1];
printf "dN,dS,dN/dS for seqs 0 and 1 is %.4f, %.4f,%.4f\n",
    $pair1->{dS}, $pair1->{dN}, $pair->{omega};
```
use Bio::Tools::Phylo::PAML;
my $outcodeml = shift(@ARGV);
my $paml_parser = Bio::Tools::Phylo::PAML->new
(-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;
        # Determine the ID should be. For leaf
        # or tip node this is just the taxon label
        if( $node->is_Leaf() ) {
          $id = $node->id;
        } else {
          # Determine the ID should be. For leaf
          # or tip node this is just the taxon label
          # or for internal nodes we need to use
          # some other method to determine the ID
        }
# Internal nodes concate names of sub-nodes
# Like how Sanderson does in r8s
$id = "\\n.join(\\n\\n, map \{ \$_->id \}
grep \{ \$_->is_Leaf \} $node->get_all_Descendents)."");
}
if( ! $node->ancestor || ! $node->has_tag( 't' ) ) {
    # skip when no values associated with this node
    next; }
printf join("\t",$id,'t=\%.3f','S=\%.1f','N=\%.1f',
    'dN/dS=\%.4f','dN=\%.4f','dS=\%.4f','S*dS=\%.1f',
    "N*dN=\%.1f\n"),
    map \{ ($node->get_tag_values($_))[0] \}
    qw(t S N dN/dS dN dS), 'S*dS', 'N*dN';
}
Workflows and Pipelines
Practical workflows

**Interchangeable parts**

- Different algorithms can be swapped
  - Similarity search: BLAST ↔ FASTA ↔ SSEARCH
  - Orthology & Paralogy: BRH ↔ InParanoid ↔ OrthoMCL
  - Gene families: Single-Linkage ↔ Jaccard Clustering ↔ MCL
  - Alignment: CLUSTALW ↔ MUSCLE ↔ T-Coffee ↔ ProbCons
  - Tree Building: Parsimony (PHYLIP) ↔ Neighbor-Joining (PHYLIP) ↔ Maximum Likelihood (PHYML, RAxML, ProtML) ↔ Bayesian (MrBayes)
  - Distances and Rates: PAML ↔ HyPHY ↔ MEGA
Build gene family

- All-vs-All similarity searches; all pairwise combos or just one big file
- OrthoMCL to build orthologous families
- Use MCL to build gene families
Practical workflows

Running OrthoMCL

$ orthomcl.pl --mode 1 --fa_files Species1.fa,Species2.fa,Species2.fa
$ cat all_orthomcl.out

ORTHOMCL5483(5 genes, 5 taxa): Afu1g13560(afum) AN1T_08052(anid) A0090012000520(aory) ATET_00564(ater) URET_04572(uree)
ORTHOMCL5484(5 genes, 5 taxa): Afu1g14430(afum) AN1T_08123(anid) A0090011000364(aory) ATET_00643(ater) HCAT_07185(hcap)
ORTHOMCL5485(5 genes, 5 taxa): Afu1g14860(afum) AN1T_06394(anid) A0090005001247(aory) ATET_00863(ater) NCUT_03527(ncra)
Running MCL

Make the all-vs-all FASTA into a tab delimited result, then run MCL on this.

$ blastall -i all.pep -d all.pep -p blastp -m 9 -e 1e-3 -o all.BLASTP.tab
$ mcxdeblast --m9 --score=e all.BLAST.tab
$ mclblastline --mcl-I=1.6 --start-assemble all.BLASTP.tab
Practical workflows

**Using FASTA instead of BLAST for MCL**

Make the all-vs-all FASTA into a tab delimited result, then run MCL on this.

```
$ fasta34_t -m 9 -d 0 -E 1e-3 -S -H all all > all.FASTA
$ perl $BIOPERL/scripts/search/fastam9_to_table all.FASTA > all.FASTA.tab
$ mcxdeblast --m9 --score=e all.FASTA.tab
$ mclblastline --mcl-I=1.6 --start-assemble all.FASTA.tab
```
Practical workflows

MCL output to families

0 Protein001
0 Protein020
0 Protein023
1 Protein003
1 Protein021
...

- Two column list of family and gene name
- Convert this into multi-fasta files, one for each family
- Convert it into a matrix with row for each family and count of family members in each species
Practical workflows

**Identify genes under positive selection**

- Build gene family; Identify orthologous (and paralogous) gene clusters among a set of genomes
- Build alignments and gene trees
- Run PAML under different models and look for selection
Practical workflows

**Find single copy mutually orthologous genes**

- InParanoid
- Cluster ortholog pairs by single-linkage and identify single copy genes
- Build alignments
- Infer gene trees by Bayesian and ML methods
- Build consensus tree from multiple genes OR
- Concatenate alignments and build consensus tree
Gene family size change

- MCL obtain gene family size counts.
- Add Pfam or other functional information to annotate the function of each gene family
- Obtain species tree with ultrametric branch lengths
- Run CAFE to identify lineage specific expansions or contractions
Practical workflows

Running CAFE

Data file: mammals.tab
Destination file: mammals.CAFE.out
Lambda value or range: 0.0020
P-value threshold: 0.01
Number of random samples: 1000
Choose methods to identify the bad branch:
- Likelihood Ratio Test
- Viterbi
- Branch Cutting
CAFE results


<table>
<thead>
<tr>
<th>P-value</th>
<th>Phylogeny Copy Number</th>
<th>Family</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0010</td>
<td>(((12 13 14) 11 (12 9 4)) 11 11)</td>
<td>ENSF00000001251</td>
<td>RHO GUANINE NUCLEOTIDE EXCHANGE FACTOR 10</td>
</tr>
<tr>
<td>0.0</td>
<td>(((11 15 21) 10 (8 7 5)) 10 8)</td>
<td>ENSF00000001658</td>
<td>EXOCYST COMPLEX COMPONENT SEC6</td>
</tr>
<tr>
<td>0.0</td>
<td>(((7 12 19) 8 (9 8 6)) 8 5)</td>
<td>ENSF00000001778</td>
<td>HIPPOCAMPUS ABUNDANT TRANSCRIPT 1</td>
</tr>
</tbody>
</table>
Futher Gene family explorations

Questions that one would ask of significant families

• For unusually large families, try and determine mode of duplication

• Look for genomic clustering of genes in same family

• Permutation test for significance of clustering

• Look at intron distribution (test for processed pseudogenes & retroposed genes)
Best Practices
Interchangeable parts

- Generic representation of applications with input and outputs allows programs to be tied together
- Generic parts allows interchanging of algorithms
- Atomize the steps so they can be distributed (and recovered if something fails)
- Separate biological logic from details of job execution
Best practices

Do I really need to run this compute?

- Precomputed data from EnsEMBL and mined in BioMart
- Treefam http://www.treelfam.org
- Families generated at Model Organism Databases
- OrthoMCL database, NCBI COGs & KOGs
- Berkeley PhyloFacts: http://phylogenomics.berkeley.edu/
Best Practises: Executing computation

- Simplify computation into steps - for parallel work or just simplicity
- Use simplest data formats when available
- Learn about tuning parameters for sensitivity, specificity, and speed
- Re-runnable Pipelines
  - Makefile driven jobs, integration with queuing software for clusters
  - Other tools for executing pipelines
- What happens in a failure or incomplete job?
Best practises

Best practises: Storing and staging data

• Flatfiles - can still be useful if used correctly
  – Don’t overload directories - Datastore::MD5 to help with this
  – Follow standards or have good reason for inventing new format
  – Can you justify custom XML format vs simple flatfile?

• RDBMS - SQL databases
  – Data modeling, adapting to changing complexity
  – Data centralization; I/O centralization too
  – Typically faster speed of query
  – Consider hybrid approach
Best practices

For more help

- http://bioperl.org - HOWTOs, API Docs, Mailing List
- http://www.nescent.org/wg_phyloinformatics/
- http://treesoft.sourceforge.net/ - NJTREE
- http://www.ensembl.org - EnsEMBL and Compara databases
Questions?
Thanks

- NESCent: Hilmar Lapp, Todd Vision
- UC Berkeley, Miller Institute
- EnsEMBL & TreeFam groups
- BioPerl developers - past and present