Phylogenetic Software

- PHYLIP, WEBPHYLIP, PhyloBLAST (large set of programs, command-line)
- PAUP (point-&-click) (MP-based)
- PUZZLE, TREE-PUZZLE, PAML, MOLPHY (ML-based)
- MrBAYES (Bayesian Methods)
- MACCLADE (MP?)
- LAMARC

Alignments

- Inputs for phylogenetic analysis usually is a multiple sequence alignment.
- Programs such as CLUSTALW, produce good alignments, but not good trees.
- Aligning according to secondary or tertiary trees are better for phylogenetic analysis.
- Which alignment method is better for which phylogenetic analysis method? OPEN!

Other Heuristics

- Branch Swapping to modify existing trees
- Quartet Puzzling: rapid tree searching

Perl: Examples

```
#!/usr/bin/perl -w
# Storing DNA in a variable, and printing it out

# First we store the DNA in a variable called $DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Next, we print the DNA onto the screen
print $DNA;

# Finally, we'll specifically tell the program to exit.
exit;  #test1.pl
```

Perl: Strings

```
#!/usr/bin/perl -w
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
# Concatenate the DNA fragments
$DNA3 = "$DNA1$DNA2";
print "Concatenation 1):\n\n$DNA3\n\n";
# An alternative way using the "dot operator":
$DNA3 = $DNA1 . $DNA2;
print "Concatenation 2):\n\n$DNA3\n\n";
# transcribe from DNA to RNA; make rev comp; print;
$RNA = $DNA3; $RNA =~ s/T/U/g;
$rev = reverse $DNA3; $rev =~ tr/AGCTacgt/TCGAtgca/;
print "$RNA\n$rev\n";
exit;  #test2.pl
```

Perl: arrays

```
#!/usr/bin/perl -w
# Read filename & remove newline from string
$protFile = <STDIN>; chomp $protFile;
# First we have to "open" the file
unless (open(PROTEINFILE, $protFile) {
   print "File $protFile does not exist"; exit;}
# Each line becomes an element of array @protein
@protein = <PROTEINFILE>;
print @protein;
# Print line #3 and number of lines
print $protein[2], "File contained ", scalar @protein,
   " lines\n";
# Close the file.
close PROTEINFILE;
exit; #test3.pl
```

Perl: subroutines

```
#!/usr/bin/perl -w
# using command line argument
$dna1 = $ARGV[0]; $dna2 = $ARGV[1];
# Call subroutine with arguments; result in $dna
$dna = addACGT($dna1, $dna2);
print "Add ACGT to $dna1 & $dna2 to get $dna\n\n";
exit;
##### addACGT: concat $dna1, $dna2, & "ACGT". #####
sub addACGT {
    my($dnaA, $dnaB) = @_; my($dnaC) = $dnaA.$dnaB;
    $dnaC .= 'ACGT';
    return $dnaC;
} #test4.pl
```

BioPerl Course

http://www.pasteur.fr/recherche/unites/sis/formation/bioperl/index.html

BioPerl Sequence Object

```
$seqobj->display_id(); # readable id of sequence
$seqobj->seq(); # string of sequence
$seqobj->subseq(5,10); # part of the sequence as a string
$seqobj->accession_number(); # if present, accession num
$seqobj->moltype(); # one of 'dna','rna','protein'
$seqobj->primary_id(); # unique id for sequence independent
                       # of its display_id or accession number
```

Sequence Formats in BioPerl

```
#! /local/bin/perl -w

use strict;
use Bio::SeqIO;
my $in  = Bio::SeqIO->newFh ( -file   => '<seqs.html',
                              -format => 'swiss' );
my $out = Bio::SeqIO->newFh ( -file   => '>seqs.fasta',
                              -format => 'fasta' );

print $out $_ while <$in>;

exit; #testx1.pl
```

BioPerl

```
#!/usr/bin/perl -w
# define a DNA sequence object with given sequence
$seq = Bio::Seq->new('-seq'=>'actgtggcgtcaact',
   '-desc'=>'Sample Bio::Seq object',
   '-display_id' => 'somethingxxx',
   '-accession_number' => 'accnumxxx',
   '-alphabet' => 'dna' );
$gb = new Bio::DB::GenBank();
# this returns a Seq object :
$seq1 = $gb->get_Seq_by_id('MUSIGHBA1');
# this returns a Seq object :
$seq2 = $gb->get_Seq_by_acc('AF303112'))
# this returns a SeqIO object :
$seqio = $gb->get_Stream_by_batch([ qw(J00522 AF303112)]));
exit; #test5.pl
```

Sequence Manipulations

```
#!/local/bin/perl -w

use Bio::DB::GenBank;
$gb = new Bio::DB::GenBank();
$seq1 = $gb->get_Seq_by_acc('AF303112');
$seq2=$seq1->trunc(1,90);
print $seq2->seq(), "\n";
$seq3=$seq2->translate;
print $seq3->seq(), "\n";
exit; #test8.pl
```

BioPerl:Download GenBank Sequences

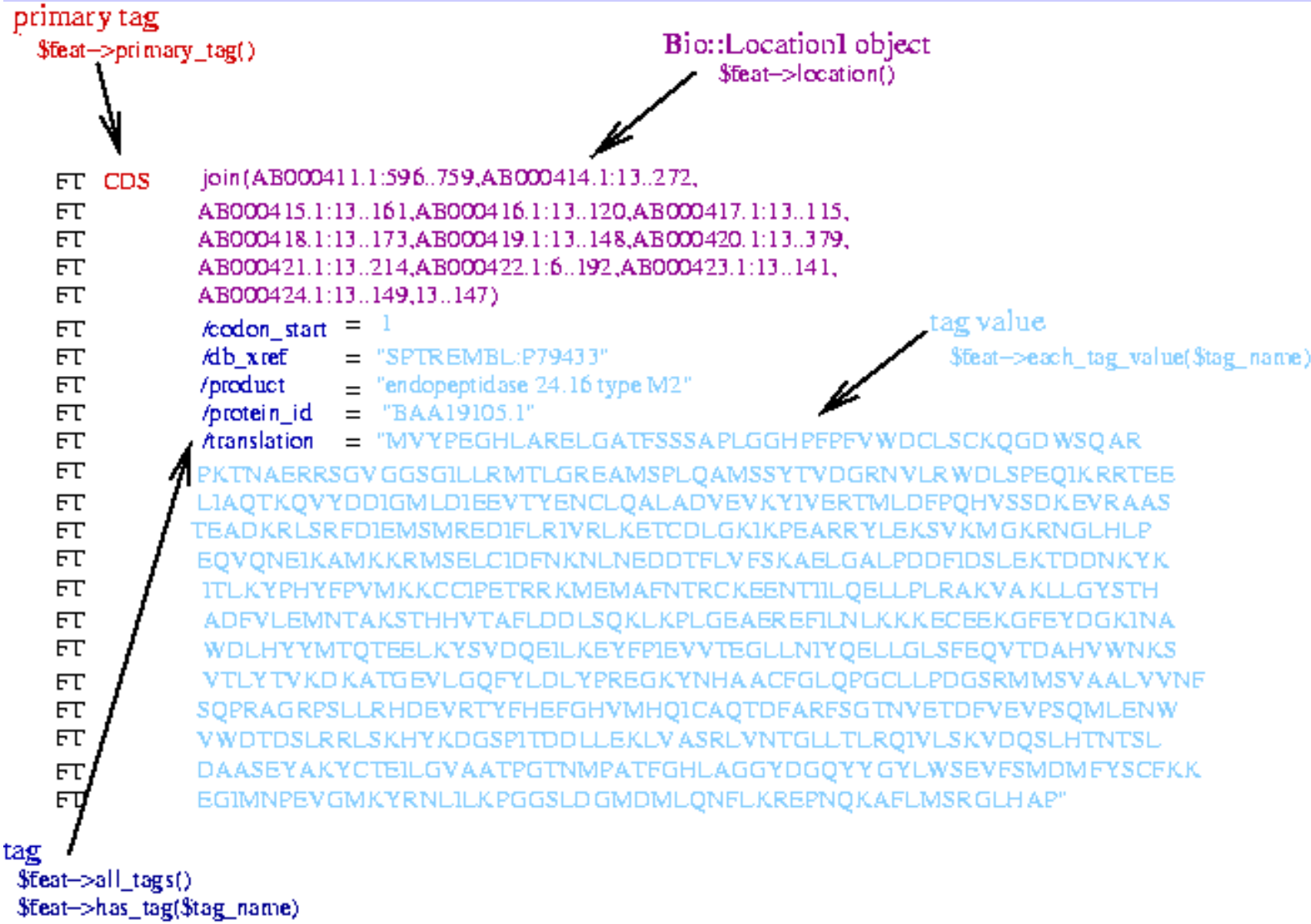
```
#!/local/bin/perl -w

use Bio::DB::GenBank;

my $gb = new Bio::DB::GenBank(
    -retrievaltype=>'tempfile', -format=>'Fasta');

my ($seq) = $seq = $gb->get_Seq_by_id("5802612");
print $seq->id, "\n";
print $seq->desc(), "Sequence: \n";
print $seq->seq(), "\n";
exit; #test6.pl
```

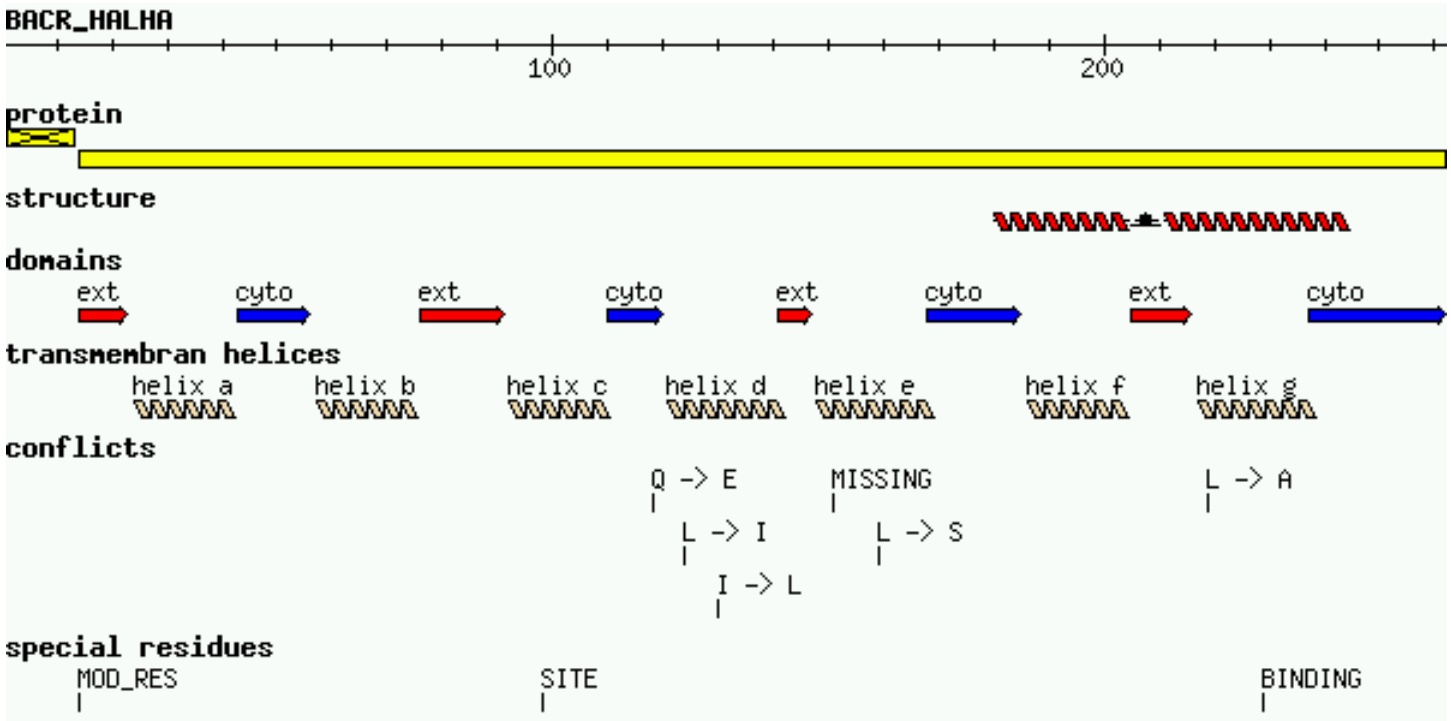
Sequence Features



BioPerl: Seq and SeqIO

```
use Bio::Seq; use Bio::SeqIO;
$seqin = Bio::SeqIO->new(-format =>'EMBL', -file=>`f1');
$seqout= Bio::SeqIO->new(-format =>'Fasta',-file=>'>f1.fa');
while((my $seqobj = $seqin->next_seq())) {
   print "Seq: ", $seqobj->display_id, ", Start of seq ",
        substr($seqobj->seq,1,10),"\n";
   if( $seqobj->moltype eq 'dna') {
        $rev = $seqobj->revcom;
        $id = $seqobj->display_id();
        $id = "$id.rev";
        $rev->display_id($id);
        $seqout->write_seq($rev); } #end if
   foreach $feat ( $seqobj->top_SeqFeatures() ) {
        if( $feat->primary_tag eq 'exon' ) {
            print STDOUT "Location ",$feat->start,":",
            $feat->end," GFF[",$feat->gff_string,"]\n";}
   } # end foreach
} # end while
exit; # test7.pl
```

BioPerl Graphics Objects



textx2.pl can create such a graphics object from a SWISS-PROT file.

BioPerl Sequence Analysis Tools

```
$seq_stats = Bio::Tools::SeqStats->new(-seq=>$seqobj);
$seq_stats->count_monomers();
$seq_stats->count_codons();
$weight = $seq_stats->get_mol_wt($seqobj);

$pat = 'T[GA]AA...TAAT';
$pattern = new Bio::Tools::SeqPattern(-SEQ =>$pat, -TYPE
    =>'Dna');
$pattern->expand;
$pattern->revcom;
$pattern->alphabet_ok;
```

BioPerl Restriction Enzymes

- Locating restriction enzyme cutting sites:
  - RestrictionEnzyme object ;
  - data for over 150 restriction enzymes built in.
  - Access list of available enzymes using available_list()
- Restriction sites can be obtained by cut_seq().
- Adding an enzyme not in the default list is easy.

Restriction Enzymes example

```
#!/local/bin/perl -w

$re=new Bio::Tools::RestrictionEnzyme('-name'=>'EcoRI');
@sixcutters = $re->available_list(6);

$re1 = new Bio::Tools::RestrictionEnzyme(-name=>'EcoRI');
# $seqobj is the Seq object for the dna to be cut
@fragments = $re1->cut_seq($seqobj);

$re2 = new Bio::Tools::RestrictionEnzyme('-NAME' =>'EcoRV--
    GAT^ATC', '-MAKE' =>'custom');

exit;
```

Alignment Object

```
#! /local/bin/perl -w
use strict;
use Bio::AlignIO;
my $inform = shift @ARGV || 'clustalw';
my $outform = shift @ARGV || 'fasta';
my $in = Bio::AlignIO->newFh ( -fh => \*STDIN,
    -format => $inform );
my $out = Bio::AlignIO->newFh ( -fh => \*STDOUT, -format =>
    $outform );

print $out $_ while <$in>;
exit;
```

Alignment Object

```
#! /local/bin/perl -w
use strict;
use Bio::AlignIO;
my $in = new Bio::AlignIO ( -file =>, $ARGV[0], -format => 'clustalw' );
my $aln = $in->next_aln();
print " all seqs same length: ",($aln->is_flush()) ? "yes" : "no", "\n";
print "alignment length: ", $aln->length(), "\n";
printf "identity: %.2f %%\n", $aln->percentage_identity();
printf "identity of conserved columns: %.2f %%\n",
    $aln->overall_percentage_identity();
```

BioPerl: Pairwise Sequence Alignment

```
use Bio::Tools::pSW;

$factory = new Bio::Tools::pSW( '-matrix' =>
   'blosum62.bla', '-gap' => 12, '-ext' => 2, );

$factory->align_and_show($seq1, $seq2, STDOUT);
```

BioPerl: Running BLAST

```
# This program only shows how to invoke BLAST and store the result
use Bio::SeqIO;
use Bio::Tools::Run::RemoteBlast;
my $Seq_in = Bio::SeqIO->new (-file => $ARGV[0], -format => 'fasta');
my $query = $Seq_in->next_seq();
my $factory = Bio::Tools::Run::RemoteBlast->new( '-prog' => 'blastp',
    '-data' => 'swissprot', _READMETHOD => "Blast" );
my $blast_report = $factory->submit_blast($query);
my $result = $blast_report->next_result;
while( my $hit = $result->next_hit()) {
    print "\thit name: ",
    $hit->name(), " significance: ", $hit->significance(), "\n";
}
# There are programs on the bioperl website that can help you automatically
    parse the information returned by BLAST.
```

BioPerl: Multiple Sequence Alignment

```
@params = ('ktuple' => 2, 'matrix' => 'BLOSUM');
$factory =
Bio::Tools::Run::Alignment::Clustalw->new(@params);
$aln = $factory->align(\@seq_array);


foreach $seq ( $aln->eachSeq() ) {
        print $seq->seq(), "\n"; }
```

BioPerl: Structure

- Ability to store and manipulate structures.
- Modules: Atom, Chain, Residue, Model, Entry, IO
- Atom
  - new, x, y, z, xyz, residue, element,
- Chain, Residue
- Entry
  - Add_model, chain, add_chain, residue, add_residue, get_residue, add_atom, get_atoms, conect, get_atom_by_serial, seqres, …
- Model

BioPerl: Structure

```
use Bio::Structure::IO;
$in = Bio::Structure::IO->new(-file => "inputfilename" , '-format' => 'pdb');
$out = Bio::Structure::IO->new(-file => ">outputfilename" , '-format' => 'pdb');
# note: we quote -format to keep older perl's from complaining.
while ( my $struc = $in->next_structure() ) {
    $out->write_structure($struc);
    print "Structure ",$struc->id," number of models: ",
            scalar $struc->model,"\n";
}
```

More Bioperl Modules

Bioperl-1.0.2::Bio::Structure::SecStr::DSSP
bioperl-1.0.2::Bio::Structure::SecStr::STRIDE
bioperl-1.0.2::Bio::Symbol
bioperl-1.0.2::Bio::Tools
bioperl-1.0.2::Bio::Tools::Alignment
bioperl-1.0.2::Bio::Tools::Bplite
bioperl-1.0.2::Bio::Tools::Blast
bioperl-1.0.2::Bio::Tools::HMMER
bioperl-1.0.2::Bio::Tools::Prediction
bioperl-1.0.2::Bio::Tools::Run::Alignment
bioperl-1.0.2::Bio::Tools::Sim4
bioperl-1.0.2::Bio::Tools::StateMachine
bioperl-1.0.2::Bio::Tree
bioperl-1.0.2::Bio::TreeIO