Master in Bioinformatica

Introduzione a BioPerl

Andrea Sterbini
sterbini@di.uniroma1.it

Installazione

- **www.bioperl.org**
- A mano

```
> tar  xzvf  bioperl-1.4.tar.gz
> cd bioperl-1.4
> perl Makefile.PL
> make
> make test
> make install
```

- Con CPAN

```
> cpan
> install Bundle::BioPerl
> install Bio::Perl
```

  - Gestisce i prerequisiti
  - Serve la connessione a Internet

- Bioperl-run, Bioperl-ext
  - PISE, EMBOSS, pSW, clustalw, NCBI-blast, TCoffee
- Bioperl-db (database)
  - Mysql, Postgres, Oracle
- Altri package
  - Bioperl-microarray, bioperl-pedigree, bioperl-gui, bioperl-pipeline, bioperl-das-client, bioperl-corba-client

Sequenze

- Seq
  - PrimarySeq
  - LocatableSeq
  - RelSegment
  - LiveSeq
  - LargeSeq
  - RichSeq
  - SeqWithQuality
  - (SeqI)
- Locations
- SeqIO

- Creazione da stringa

```
use Bio::Seq;
$seq = Bio::Seq -> new(
    -seq                => 'actgtggcgtcaact',
    -desc               => 'Esempio di  Bio::Seq',
    -display_id         => 'qualcosa',
    -accession_number   => 'accnum',
    -alphabet           => 'dna' );
```

- I/O da file e trasformazione di formato

```
use Bio::SeqIO;
$in  = Bio::SeqIO -> new(
        -file => "inputfilename",
        -format => 'Fasta');
$out = Bio::SeqIO -> new(
        -file => ">outputfilename",
        -format => 'EMBL');
while ( my $seq = $in -> next_seq() ) {
        $out->write_seq($seq); }
```

Input da database in rete

- Database in rete
  - Genbank
    - NCBI
  - Genpept
  - RefSeq
  - Swissprot
  - EMBL
- Ma anche
  - Ace
  - SWALL

- Collegamento al database Genbank

```
use Bio::DB::GenBank;
$gb = new Bio::DB::GenBank();
```

- Scaricare una sequenza conoscendone lo ID

```
$seq1 = $gb->get_Seq_by_id(
    'MUSIGHBA1' );
```

- Scaricare una sequenza conoscendone l'Accession code

```
$seq2 = $gb->get_Seq_by_acc(
    'AF303112' );
```

- Apertura di uno stream su tre sequenze

```
$seqio = $gb-> get_Stream_by_id(
    ["J00522", "AF303112", "2981014"]);
```

Sequenze ...

- ID della sequenza

```
$seqobj->display_id();
```

- Stringa della sequenza

```
$seqobj->seq();
```

- Sottosequenza

```
$seqobj->subseq(5,10);
```

- Accession number

```
$seqobj -> accession_number();
```

- Alfabeto 'dna', 'rna', 'protein'

```
$seqobj->alphabet();
```

- ID unico

```
$seqobj->primary_id();
```

- Da 'dna' a 'protein'

```
$translation1 = $seqobj->translate;
```

- Parametri aggiuntivi:
  - frame (0, 1 o 2)

```
$seqobj -> translate( undef, undef, 1);
```

  - codon table (sono 16)

```
$seqobj -> translate( undef, undef,
    undef, 2);
```

  - Full coding regions

```
$cds -> translate( undef, undef, undef,
    undef, 1, 'die_if_errors');
```

... e ancora Sequenze

- Statistiche

```
use Bio::Tools::SeqStats;
$stats  = Bio::Tools::SeqStats ->
new($seqobj);
```

- Peso molecolare

```
$seq_stats ->get_mol_wt();
```

- Numero di monomeri [min, max]

```
$seq_stats -> count_monomers();
```

- Numero di codoni (per dna, rna)

```
$codon_ref = $seq_stats -> count_codons();
```

- Aminoacidi 'carichi'

```
use Bio::Tools::OddCodes;
$oddcode_obj = Bio::Tools::OddCodes ->
new($amino_obj);
$output = $oddcode_obj -> charge();
```

- Descrizione chimica

```
$output = $oddcode_obj->chemical();
```

  - 'ACDEFGH' diventa 'LSAARAC'

- **A (acido), L (alifatico), M (amide), R (aromatico), C (basico), H (idrossile), I (imino), S (zolfo)**

Restriction enzymes

- Più di 500 enzimi di tipo II

```
use Bio::Restriction::EnzymeCollection;
my $all_collection = Bio::Restriction::EnzymeCollection;
```

- Come selezionare quelli lunghi 6 basi

```
my $six_cutter_collection = $all_collection -> cutters(6);
```

- Selezione di un enzima per nome

```
my $ecori_enzyme = $all_collection -> get_enzyme('EcoRI');
```

- Estrazione dei siti di taglio (dà una lista di stringhe)

```
use Bio::Restriction::Analysis;
my $analysis = Bio::Restriction::Analysis -> new(-seq => $seq);
@fragments =  $analysis -> fragments($enzyme);
```

- Creazione di un nuovo restriction enzyme

```
my $re = new Bio::Restriction::Enzyme(-enzyme=>'BioRI', -seq=>'GG^AATTCC');
```

Blast in remoto

- Usando Blast via rete su NCBI (con tempi elevati)

```
$remote_blast =
Bio::Tools::Run::RemoteBlast -> new (
        -prog => 'blastp', -data => 'ecoli',
        -expect => '1e-10' );
$r = $remote_blast -> submit_blast(
        "t/data/ecolitst.fa");
while (@rids = $remote_blast -> each_rid ) {
    foreach $rid ( @rids ) {
        $rc = $remote_blast ->
                retrieve_blast($rid);
} }
```

- Esaminare i risultati con BIO::SearchIO

```
$searchio = new Bio::SearchIO (
    -format => 'blast', -file   => $blast_report );
$result = $searchio->next_result;
```

- Informazioni globali

```
$result->database_name;
$alg_type =  $result -> algorithm;
```

- Informazioni sulla prima corrispondenza trovata

```
$hit = $result->next_hit;
$hit_name = $hit->name ;
```

- Informazioni sulla prima HSP della corrispondenza trovata

```
$hsp = $hit->next_hsp;
$hsp_start = $hsp->query->start;
```

Blast locale

- Deve essere installato Blast
  - Deve essere definite le variabili BLASTDIR, BLASTDB
  - Creo un wrapper StandAloneBlast

```
use Bio::Tools::Run::StandAloneBlast;
@params = (program  => 'blastn', database => 'ecoli.nt');
$factory = Bio::Tools::Run::StandAloneBlast->new(@params);
```

  - Eseguo la ricerca

```
$input = Bio::Seq->new(-id  =>"test query", -seq =>"ACTAAGTGGGGG");
$blast_report = $factory->blastall($input);
```

  - Esamino il report

```
while (my $hit = $blast_report->next_hit){
    while (my $hsp = $hit->next_hsp){
        print $hsp->score," ",$hit->name,"\n";
} }
```

Ricerca di Geni

- Interfaccia ai dati di:
  - Genscan, Sim4, Genemark, Grail, ESTScan, MZEF, ePCR
- Lettura di report Genscan

```
use Bio::Tools::Genscan;
$genscan = Bio::Tools::Genscan ->
    new(-file => 'result.genscan');
# $gene è un Bio::Tools::Prediction::Gene
# $gene->exons() torna un array di oggetti
# di tipo Bio::Tools::Prediction::Exon
while($gene = $genscan ->next_prediction())
{
    @exon_arr = $gene->exons();
}
$genscan->close();
```

- Lettura di report Sim4

```
use Bio::Tools::Sim4::Results;
$sim4 = new Bio::Tools::Sim4::Result(
    -file=>'t/data/sim4.rev',
    -estisfirst =>0);
# creo un Bio::SeqFeature::Generic con
# Bio::Tools::Sim4::Exons
# come sub-features
$exonset = $sim4->next_exonset;
@exons=$exonset->sub_SeqFeature();
# $exon è Bio::SeqFeature::FeaturePair
$exon = 1;
$exonstart = $exons[$exon]->start();
$estname = $exons[$exon]->est_hit()
                        ->seqname();
$sim4->close();
```

Allineamenti

- Input da:
  - bl2seq, clustalw, emboss, fasta, mase, mega, meme, metafasta, msf, nexus, pfam, phylip, prodom, psi, selex, stockholm
- 6 formati di output:
  - fasta, mase, selex, clustalw, msf/gcg, phylip (interleaved)
- Esempio di I/O e trasformazione di formato

```
use Bio::AlignIO;
$in   = Bio::AlignIO -> new(   -file => "inputfilename",   -format => 'pfam');
$out = Bio::AlignIO -> new(   -file => ">outputfilename",-format => 'fasta');
while ( my $aln = $in->next_aln() ){
    $out->write_aln($aln);
}
```

Patterns e coordinate

- Manipolazione di sequenze con espressioni regolari

```
use Bio::Tools::SeqPattern;
$pattern     = '(CCCCT)N{1,200}(agggg)N{1,200}(agggg)';
$pattern_obj = new Bio::Tools::SeqPattern(-SEQ  => $pattern, -TYPE => 'dna');
$pattern_obj2 = $pattern_obj -> revcom();
$pattern_obj -> revcom(1); # torna il pattern complementare inverso esteso
```

- Conversione di coordinate

```
$input_coordinates = Bio::Location::Simple -> new(
    -seq_id => 'propeptide', -start => 1000, -end => 2000, -strand=>1 );
$output_coordinates = Bio::Location::Simple -> new(
    -seq_id => 'peptide', -start => 1100, -end => 2100, -strand=>1 );
$pair = Bio::Coordinate::Pair -> new(
    -in => $input_coordinates , -out => $output_coordinates   );
$pos = Bio::Location::Simple -> new (-start => 500, -end => 500 );
$res = $pair->map($pos);
$converted_start = $res->start;
```

bptutorial.pl

- Il BioPerl tutorial
  è un programma Perl!
  - 25 esempi di uso di BioPerl
  - Documentazione (estraibile con perldoc)
  - Usa alcuni files della distribuzione (va eseguito sul posto)
- Installazione:

  > tar xzvf bioperl-1.4.tar.gz

  > cd bioperl-1.4

- Pagina di help

  > perl -w bptutorial.pl

- Esecuzione di una demo

  > perl -w bptutorial.pl N

- Metodi di un modulo (anche quelli ereditati)

  > perl -w bptutorial.pl 100 Bio::Seq

- Tutorial in formato testo

  > perldoc bptutorial.pl

Segue ...

Accidenti ai proxy! :-)

- Per attraversare il proxy scrivete:

  > export http_proxy='http://10.0.0.100:3128'

- A questo punto potete provare i tutorial 11 e 22
  - Esempio di accesso a database NCBI

  > perl bptutorial.pl 11

  - Esempio di accesso a BLAST remoto

  > perl bptutorial.pl 22

- Gli ultimi 3 tutorial necessitano di programmi esterni:
  - blast, clustalw e tcoffee, psw e bl2seq

Sequence Annotations

- Annotazione associata ad una posizione, contiene:

```
$feat->start            # inizio
$feat->end              # fine
$feat->strand           # 1 vuol dire 'in avanti', -1 'indietro', 0 non rilevante
$feat->primary_tag      # 'nome' della feature, ad esempio 'exon'
$feat->source_tag       # sorgente della feature, ad esempio 'BLAST'

# metodi che tornano un oggetto Bio::PrimarySeq
$feat->seq              # la sequenza compresa tra 'start' ed 'end'
$feat->entire_seq       # la sequenza intera
$feat->spliced_seq      # la sequenza “unita”, quando ci sono posizioni multiple

$feat->overlap($other)  # le due feature $feat e $other si sovrappongono?
$feat->contains($other) # la feature $other è completamente dentro a $feat?
$feat->equals($other)   # $feat e $other sono esattamente uguali?
$feat->sub_SeqFeatures  # crea/legge un array di feature di sottosequenze
```

- Annotazioni dettagliate (RichSeq) ricavate da DB

```
@secondary    = $richseq->get_secondary_accessions;
$division     = $richseq->division;
@dates        = $richseq->get_dates;
$seq_version  = $richseq->seq_version;
```

Sequenze enormi (LargeSeq)

- Per manipolare sequenze enormi (>100 Mbasi)

```
$seqio = new Bio::SeqIO(
                        -format => 'largefasta',
                        -file   => 't/data/genomic-seq.fasta');
$pseq      = $seqio->next_seq();
$plength   = $pseq->length();
$last_4    = $pseq->subseq($plength-3,$plength);      # OK
```

- Si possono usare tutti i metodi di una Seq tranne quelli che tornano come risultato tutta la sequenza

```
# questo dà quasi sicuramente un errore di memoria!!!

# $lots_of_data = $pseq->seq();  # NON OK per una LargeSeq
```

Sequenze che cambiano (LiveSeq)

- Con le LiveSeq è possibile manipolare sequenze i cui dati vengono aggiornati frequentemente
  - Le posizioni non sono implementate come numeri costanti ma calcolate solo al momento in cui servono

```
$loader = Bio::LiveSeq::IO::BioPerl->load(
        -db  => "EMBL",
        -file => "t/data/factor7.embl");
$gene  = $loader->gene2liveseq(-gene_name => "factor7");
$id = $gene->get_DNA->display_id ;
$maxstart = $gene->maxtranscript->start;
```

Mutazioni e Polimorfismi

- 'Mutation' descrive una modifica da applicare ad una sequenza di DNA di un gene
- 'Mutator' applica le mutazioni ad un gene di una LiveSeq
- Il risultato è una lista di 'Variation' che descrivono le modifiche avvenute

- SeqDiff: differenze tra due geni

```
use Bio::LiveSeq::IO::BioPerl;
use Bio::LiveSeq::Mutator;
use Bio::LiveSeq::Mutation;
$loader = Bio::LiveSeq::IO::BioPerl->load(
        -file => "$filename");
$gene = $loader->gene2liveseq(
        -gene_name => $gene_name);
$mutation = new Bio::LiveSeq::Mutation (
        -seq =>'G', -pos => 100 );
$mutate = Bio::LiveSeq::Mutator->new(
        -gene => $gene, -numbering => "coding"  );
$mutate->add_Mutation($mutation);
$seqdiff = $mutate->change_gene();
$DNA_re_changes = $seqdiff -> DNAMutation ->
restriction_changes;
$RNA_re_changes = $seqdiff -> RNAChange ->
restriction_changes;
if (! $DNA_re_changes eq $RNA_re_changes) {
    print "Different!\n"
};
```

Clusters di sequenze (Cluster, ClusterIO)

- Manipolazione di insiemi di sequenze
  - E' previsto l'I/O solo in formato Unigene
- Esempio: estrarre le sequenza da un cluster e stamparne gli accession numers

```
my $stream = Bio::ClusterIO->new(
        -file => "Hs.data", -format => "unigene");
while ( my $in = $stream->next_cluster ) {
    print $in->unigene_id() . "\n";
    while ( my $sequence = $in->next_seq ) {
        print $sequence->accession_number . "\n";
    }
}
```

Strutture 3D e files PDB (StructureIO)

- Operazioni sui dati di strutture 3D
  - Leggo un file PDB

```
$structio = Bio::Structure::IO->new(
                    -file => "1XYZ.pdb");
$struc    = $structio->next_structure;   # torna una Entry
$pseq     = $struc->seqres;              # torna una PrimarySeq
$pseq->subseq(1,20);                     # torna una stringa
```

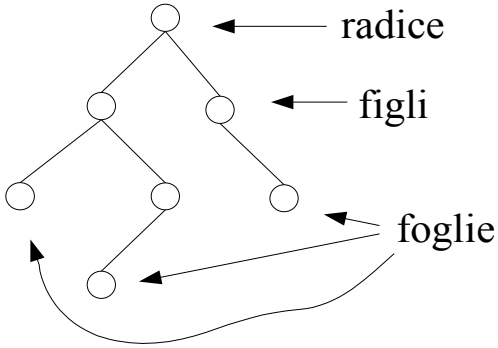
  - Estrazione delle coordinate degli atomi di un Residuo

```
# dato un Residue, torna una lista di oggetti Atom
@atoms  = $struc->get_atoms($res);
@xyz    = $atom->xyz;    # le coordinate 3D dell'atomo
```

Alberi e alberi filogenetici

- Potete manipolare alberi generici con la classe Tree
  - In informatica hanno radice in alto e foglie in basso :-)
  - I 'figli' di un nodo sono i nodi immediatamente sotto
  - Le foglie sono nodi che non hanno nessun figlio
- TreeIO per leggere/scrivere i formati phylip o newick

```
$treeio     = new Bio::TreeIO( -format => 'newick', -file => $treefile);
$tree       = $treeio->next_tree;          # legge un albero dal file
@nodes      = $tree->get_nodes;            # torna tutti i nodi
# ottiene tutti i disecendenti del nodo radice
$tree->get_root_node->each_Descendent;
```



- Con PAML leggete l'output di:
  - codeml, baseml, basemlg, codemlsites e yn00

Programmi esterni: EMBOSS

- Con bioperl-ext potete usare i programmi EMBOSS
  - Esempio:

```
$factory = new Bio::Factory::EMBOSS;
$compseqapp = $factory->program('compseq');
%params = (   -word=>4, -sequence=>$seqObj,
              -outfile=>$compseqoutfile );
$compseqapp->run(\%params);
$seqio = Bio::SeqIO->new( -file => $compseqoutfile );
```

  - Si possono usare anche nomi di file per l'input e l'output:

```
-sequence => "inputfasta.fa"
```

Programmi esterni: Pise

- Se non avete EMBOSS installato potete usare i programma installati sul server Pise (Pasteur Institute)

```
my $genscan = Pise::genscan->new(
    "http://bioweb.pasteur.fr/cgi-bin/seqanal/genscan.pl",
    'letondal@pasteur.fr',
    seq => $seq,    parameter_file => "Arabidopsis.smat");
 my $job = $genscan->run;
 my $parser = Bio::Tools::Genscan -> new(
                        -fh => $job->fh('genscan.out') );
 while(my $gene = $parser->next_prediction()) {
    my $prot = $gene->predicted_protein;
    print $prot->seq, "\n";
 }
```

Allineare due sequenze

- Potete usare bl2seq e AlignIO per allineare due sequenze usando blast in locale
  - Create un lanciatore di blast in locale

```
$factory = Bio::Tools::Run::StandAloneBlast ->
   new('outfile' => 'bl2seq.out');
```

  - Eseguite bl2seq e ne ricavate il report

```
$bl2seq_report = $factory->bl2seq($seq1, $seq2);
```

  - Con AlignIO create un SimpleAlign dai dati del report

```
$str = Bio::AlignIO->new(
        -file   => 'bl2seq.out', -format => 'bl2seq');
$aln = $str->next_aln();
```

Allineare sequenze multiple

- Per sequenze multiple si può usare clustalw o tcoffee
  - Devono essere installati nel sistema
  - Vanno settate CLUSTALDIR e TCOFFEEDIR

```
use Bio::Tools::Run::Alignment::Clustalw;
@params = ('ktuple' => 2, 'matrix' => 'BLOSUM');
$factory = Bio::Tools::Run::Alignment::Clustalw ->
                    new(@params);
$factory->ktuple( 3 );
$seq_array_ref = \@seq_array;
# dove @seq_array è un vettore di oggetti Bio::Seq
$aln = $factory -> align($seq_array_ref);
```

Allineamenti con Smith-Waterman

- Deve essere installato bioperl-ext E pSW
- Solo per proteine
- Sono presenti le matrici blosum62 e gonnet250
- Se ne possono aggiungere altre
- Esempio

```
use Bio::Tools::pSW;
$factory = new Bio::Tools::pSW(
        '-matrix' => 'blosum62.bla', '-gap' => 12, '-ext' => 2, );
$factory->align_and_show($seq1, $seq2, STDOUT);
$aln = $factory->pairwise_alignment($seq1, $seq2);
```