CMSC423: Bioinformatic Algorithms, Databases and Tools

Writing bioinformatics software

Libraries & misc.

Libraries/utilities

- Bio::Perl (Perl)
- BioJava (Java)
- BioPython (Python)
- BioRuby (Ruby)
- seqAn (C++)
- Bioconductor (R)
- Chado (SQL)

Bio::Perl

- http://www.bioperl.org

```
use Bio::Perl;

my $seq = read_sequence(“mytest.fa”, “fasta”);
my $gbseq = read_sequence(“mytest.gb”, “genbank”);

write_sequence(“>test.fasta”, 'fasta', $gbseq);
```

' vs “ ?

Bio::Perl

- Find sequences longer than 500 letters

```
use Bio:Perl;


while ($seq = read_sequence(“test.fa”, 'fasta')) {
    if ($seq ->length() > 500) {
        print $seq->primary_id(), “\n”;
    }
}
```

Bio::Perl

- Other useful stuff

```
$seqio = new Bio::SeqIO(-format => 'largefasta',
                        -file   => 't/data/genomic-seq.fasta');
$pseq = $seqio->next_seq();

$gb = new Bio::DB::GenBank;
$seq1 = $gb->get_Seq_by_id('MUSIGHBA1');
```

etc...

BioJava

- http://www.biojava.org

```
import org.biojava.bio.*;
String filename = args[0];
    BufferedInputStream is =
        new BufferedInputStream(new FileInputStream(filename));
    //get the appropriate Alphabet
    Alphabet alpha =
  AlphabetManager.alphabetForName(args[1]);

    //get a SequenceDB of all sequences in the file
    SequenceDB db = SeqIOTools.readFasta(is, alpha);
```

BioJava

```
BufferedReader br =
        new BufferedReader(new FileReader(args[0]));

 String format = args[1];
 String alphabet = args[2];

 SequenceIterator iter =
 (SequenceIterator)SeqIOTools.fileToBiojava(format,alphabet, br);
while (iter.hasNext()){
    Sequence seq = iter.nextSequence();
    if (seq.length() > 500) {System.out.println(seq.getName());}
}
```

BioJava...more

- Same as Bio::Perl:
  - can directly connect to databases
  - various sequence manipulations (reverse complement, translate, etc.)
  - basic bioinformatics algorithms
  - etc.

BioPython

- http://www.biopython.org

```
from Bio import SeqIO
handle = open(“file.fasta”)
seq_record = SeqIO.parse(handle, “fasta”)

SeqIO.write(my_records, handle2, "fasta")
```

BioPython

```
from Bio import SeqIO
handle = open("test.fasta")
for seq_record in SeqIO.parse(handle, "fasta") :
    if len(seq_record) > 500 :
        print seq_record.id
handle.close()
```

BioPython...more

- Same as Bio::Perl:
  - can directly connect to databases
  - various sequence manipulations (reverse complement, translate, etc.)
  - basic bioinformatics algorithms
  - etc.

BioRuby

- http://www.bioruby.org

```
require 'bio'

input_seq = ARGF.read      # reads all files inarguments

my_naseq = Bio::Sequence::NA.new(input_seq)
```

BioRuby

```
#!/usr/bin/env ruby

require 'bio'

ff = Bio::FlatFile.new(Bio::FastaFormat, ARGF)
ff.each_entry do |f|
  if f.length > 500
    puts f.entry_id
  end
end
```

BioRuby...more

- Same as Bio::Perl:
  - can directly connect to databases
  - various sequence manipulations (reverse complement, translate, etc.)
  - basic bioinformatics algorithms
  - etc.

SeqAn

- http://www.seqan.de

```
#include <seqan/sequence.h>
#include <seqan/file.h>

using namespace seqan;
using namespace std;

String <Dna> seq;
String<char> name;
fstream f;
f.open(“test.fasta”);
readMeta(f, name, Fasta());
readMeta(f, seq, Fasta());
```

SeqAn

```
String <Dna> seq;
String<char> name;
fstream f;
f.open(“test.fasta”);
while (! f.eof()){
    readMeta(f, name, Fasta());
    readMeta(f, seq, Fasta());
    if (length(seq)){
        cout << name << endl;
    }
}
```

SeqAn...more

- Not quite as much as Perl/Java/Python, but still lots of utilities (including graph algorithms)

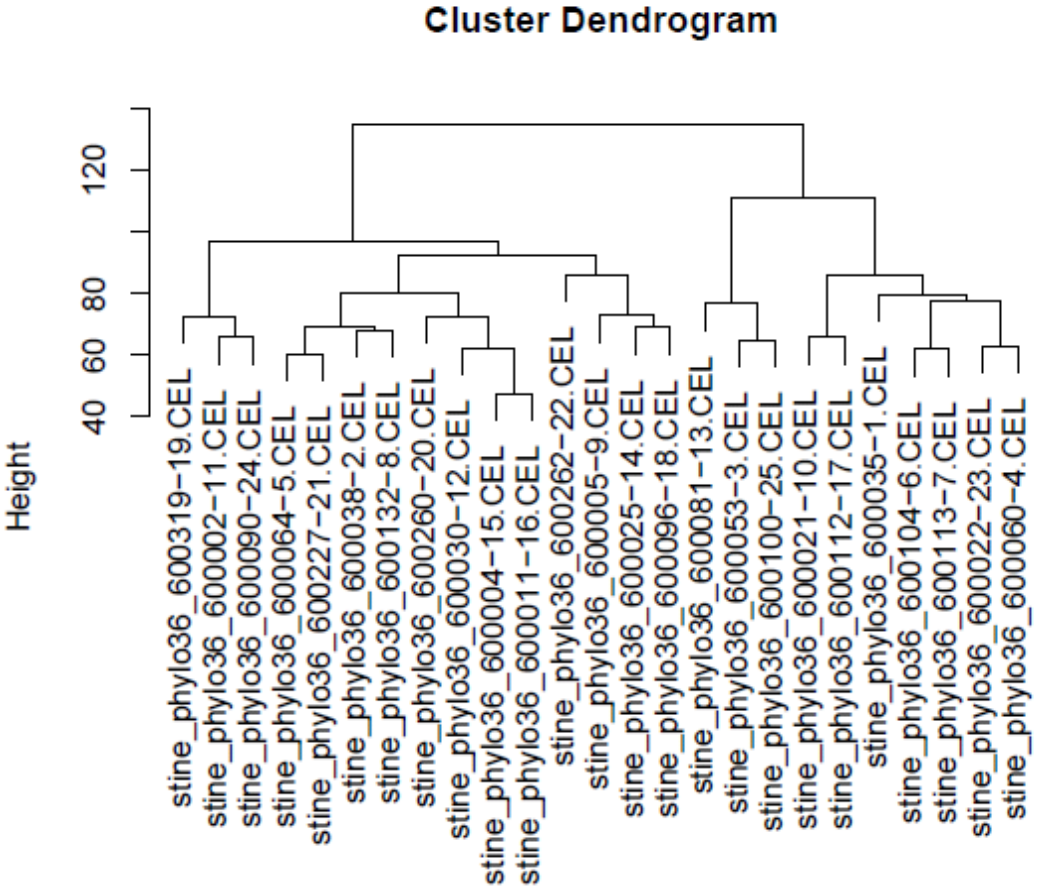
R/BioConductor

- http://www.bioconductor.org
- Mainly for statistical applications, e.g. microarray analysis

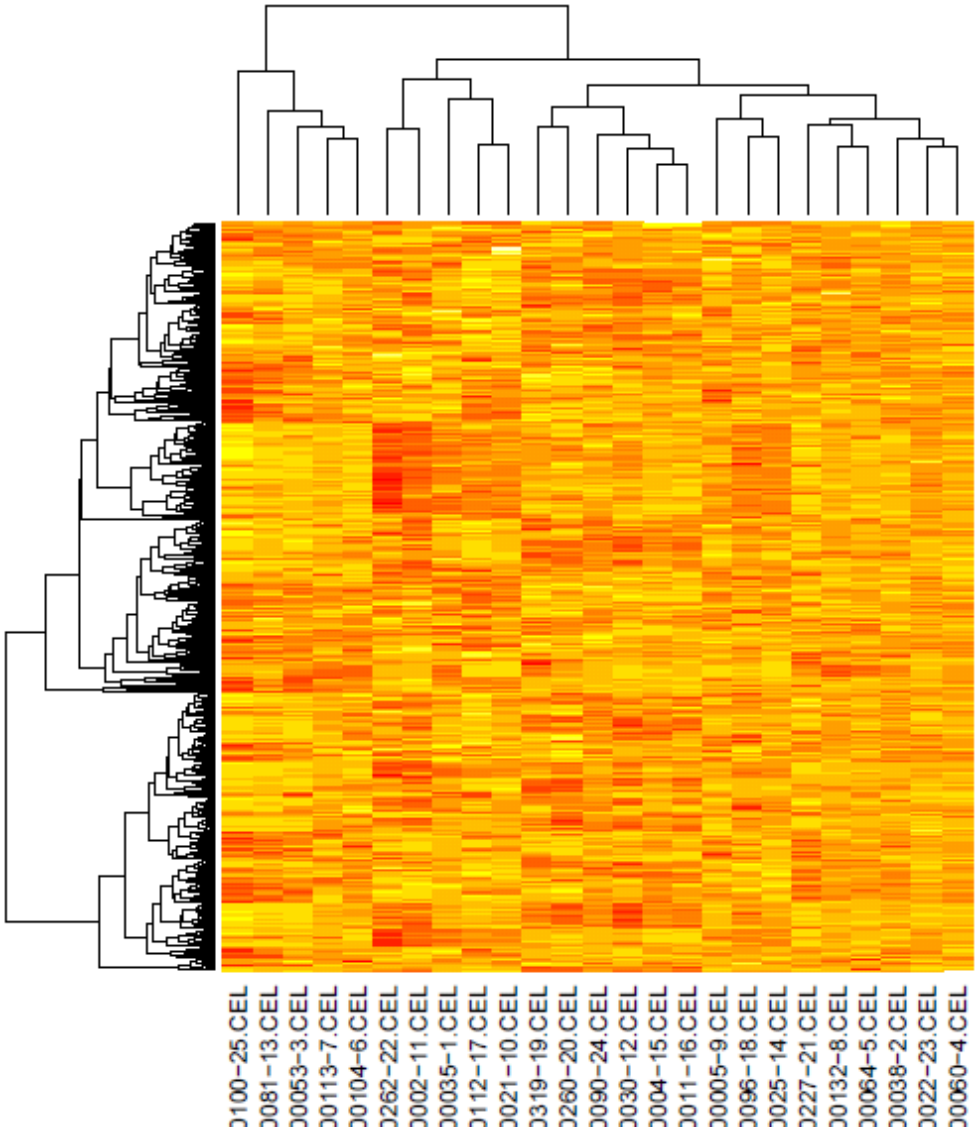
```
library("affy")
library("geneplotter")
library("gplots")


data <- ReadAffy()
eset <- rma(data)
e <- exprs(eset)
heatmap.2(e, margin=c(15,15), trace="none",
   col=redgreen(25), cexRow=0.5)
```

R/BioConductor

- Book has lots of examples
- Worth learning more about it – easy to do various cool things



R... more cool stuff



Programming for bioinformatics

- Details of specialized libraries beyond scope of course
- Good software engineering practices are essential
- Often, “correct” is undefined – output of program defines correctness
- Pitfalls – e.g. papers retracted from Science due to software bugs

- Key – be proactive and learn by yourselves from online resources!