Using Biopython for Laboratory Analysis Pipelines

Brad Chapman

27 June 2003

What is Biopython?

Official blurb – The Biopython Project is an international association of developers of freely available Python tools for computational molecular biology.

Simple terms – Develop reusable Python code for anything related to biology and computing.

Practical Things

- Established in 1999
- http://www.biopython.org
- Members of the Open Bioinformatics Foundation
- Active mailing lists; semi-regular releases; CVS. . . everything you like in an open-source project

What exactly is Python?

Official blurb – Python is an interpreted, interactive, object-oriented programming language.

My take – Scripting language with a clear syntax that is easy to learn and become productive with.

Bonus points – portable, tons of useful libraries, excellent user community, fast numerical capabilities

`http://www.python.org`

Python example

```
import random
class LotteryNumber:
    """Play LOTTO SOUTH, y'all. It's a Southern thing.
    """
    def __str__(self):
        numbers = self.pick_numbers()
        return "-".join(numbers)
    def pick_numbers(self, number = 6, low = 1, high = 49):
        values = []
        for num in range(number):
            values.append(str(random.randrange(low, high)))
        return values

winning_number = LotteryNumber()
print winning_number
```

Python one-liners

Not quite as famous as Perl, but you can jam some productive code onto a single line

Python code to generate a random sequence:

```
2:18am chapmanb> python
Python 2.2.3 (#1, Jun  5 2003, 15:04:28)
[GCC 2.95.4 20020320 [FreeBSD]] on freebsd4
Type ‘‘help’’, ‘‘copyright’’, ‘‘credits’’ or ‘‘license’’ for more
information.
>>> import random; ’’.join([random.choice(’AGTC’) for i in range(250)])
’AGAGACCCAAGATTCTTAAAAAAATGACTATAACGCGACCCATCAGATTCGCCTGTCCCCACCGCGGTGA
AATTGTCTATAGTCGTTAACCCATCTTGACGGGGCCACACTAAACTCGCAAAATTAATGAAATAAACGAGA
CAGTTGCTAACGTTCCCAGCCCGCTAGAGACTTGGCCGTCACCTAGCCGAAGATGAGTAAGGAAAGATGCC
AATCTTGGGTAGCCCTACCCACTGTAGCTGAGTCCTTA’
```

What does Biopython have for you?

Sequence-type Data – Sequences, features on sequences, storage in SQL databases (BioSQL), retrieval from on-line databases (NCBI)

Biological File Formats – FASTA, GenBank

Manipulating Sequences – Transcription, translation, slicing sequences

Bioinformatics programs – Dealing with BLAST, Clustalw, EMBOSS

Structure Data – PDB parsing, representation and analysis

Microarray Data – Clustering data, reading and writing

Substitution Matrices – Creating, manipulating, common matrices

Biopython development philosophies

Basically the philosophy of all of the various Open-Bio projects

Approach – Focus on the development of libraries, as opposed to ready-to-run programs.

Alternative Approaches

- Design one program which does something well (BLAST, Clustalw).
- Design a suite of programs that can work together (EMBOSS, PHYLIP, HMMER).

Different approach to solve problems – both have advantages and disadvantages.

Advantages of this approach

- Flexible set of "small as possible" modules that can be used in numerous ways.
- Different parts can interoperate (no reading and writing of file formats).
- Can utilize the power of a programming language.
- Automation – a single program (written by you) can accomplish several different tasks.
- Allows you to take advantage of existing programs by executing them in your program.

Disadvantages of the approach

- Need to be able to program.
- Division of labor – multiple projects for different programming languages (BioPerl, BioJava, Biopython, BioRuby. . .)
- Requires more activation energy on the part of users
  - Tough to get started and get something happening without thinking.
  - Need to learn the code base and the way things are done.

Why is this approach useful?

Laboratory Analysis Pipelines

My Definition – Set of interconnected analysis which analyze and transform data into a form useful for asking biological questions.

Why has this sort of analysis become more common?

- Growth of data – more sequences, but all kinds of data available (microarray, structures. . .)
- Interconnectedness of questions – Systems Biology.
- Can ask questions incorporating data from beyond what you generate in your own laboratory.

Examples of analysis pipelines

Align protein sequences, do bootstrapping, find distance matrices, create phylogenetic trees using neighbor joining.

Retrieve all sequences matching an Entrez query, perform a Fasta search against a local databases, retrieve those matching sequences and determine their locations on a genetic map.

Get microarray results matching a desired expression pattern, retrieve the upstream regions of the corresponding sequences, and search for conserved motifs.

Take sequences generated in the lab, process with Phred/Phrap/Crossmatch, find BLAST hits, find predicted Pfam domains, place into a database for easy display.

Common features of an analysis pipeline

Multiple data sources – Different types of data that need integration. Data not readily interchangeable.

Data widely distributed – Data may be anywhere from local databases to disparate web interfaces.

Multiple transformation steps – Data is queried and filtered through a number of different analyses.

Integrates many programs – Often make use of many standard bioinformatics programs.

Lots of data – Don't want to do it by hand.

Analysis versus Integration

Asking different types of questions then "traditional" bioinformatics when developing analysis pipelines.

Analysis

- Single individual problem
- Specialized, technical solutions
- Algorithms

Integration

- Multiple analyses
- Connect data and tools
- Queries over multiple types of data

Inspired by a CSB2002 paper by Parker, Gorlick and Lee –

http://www.bioinformatics.ucla.edu/leelab/db/large-1.15.2.pdf

Common problems and Biopython solutions

- Difficult to deal with file formats that need parsing
  - Generalized parser generator interface which makes creating parsers simpler.
    - Martel
    - Build parsers with "regular expressions on steroids"
    - Transform to XML which can be parsed with standard parsers.
- Lots of different file formats and representations of the same data (GenBank, EMBL. . .)
  - General sequence and feature classes, plus format specific classes.
  - Standard Martel XML generated tags for general items of interest.

More problems and solutions

- Multiple analysis programs with similar general outputs (BLAST, Fasta, Hmmpfam. . .).
  - Generalize most important outputs into standard formats (Martel and XML).
- Dealing with reading and writing files (managing data).
  - BioSQL interfaces for storing data in relational databases

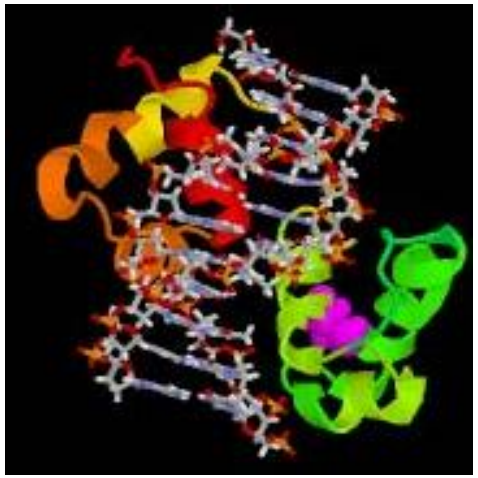
Man, there are a lot of problems

- Data items distributed in multiple locations, which change over time
  - Open Bioinformatics Database Access (OBDA) – generalized access to retrieving information into standard representations
  - EUtils at NCBI – connect with specifically designed retrieval interfaces (instead of scraping web pages).

- Massaging and changing data in potentially simple ways (translation of DNA sequences to protein)
  - Simple sequence manipulation interfaces
  - Changes which can be made through a programming language (changing titles, manipulating strings, ugly things).

Analysis pipeline example

- Myb family of transcription factors.
- Idea – look at the effect they might play during defense responses in plants.
- Heterogeneous data
  - Querying NCBI's Entrez interface for Myb genes
  - Parsing GenBank data format
  - Microarray – expression time course simulating fungal attack
- Exploratory study to determine if there are any interesting patterns of Myb gene expression.



Approach

1. Query NCBI to get a list of Myb genes in Arabidopsis

2. Download Myb genes and retrieve their Arabidopsis gene names from GenBank formatted files.

3. Load data from microarray experiment

4. Pull out Myb genes present on the microarray

5. Do clustering of the Myb gene expression patterns to look for conserved patterns of expression.

Accomplished in just over 30 lines of code using Python and Biopython libraries.

Querying Entrez at NCBI

Design a search to get Myb genes in Arabidopsis

```
search_string = "Myb AND txid3702[ORGN] AND 0:6000[SLEN]"
```

Use Biopython interface to EUtils at NCBI to do an Entrez search

```
from Bio.EUtils import HistoryClient

client = HistoryClient.HistoryClient()
results = client.search(search_string, db = "nucleotide")
```

Retrieving the results

Retrieve the search results in GenBank format using EUtils

```
gb_handle = results.efetch(retmode = "text", rettype = "gb")
```

Want to retrieve the gene name (At5g59780) from the GenBank file

```
LOCUS       At5g59780                972 bp    mRNA    linear   PLN 05-JUN-2003
DEFINITION  Arabidopsis thaliana myb family transcription factor (At5g59780)
            mRNA, complete cds.
ACCESSION   NM_125370
VERSION     NM_125370.2  GI:30697278
KEYWORDS    .
SOURCE      Arabidopsis thaliana (thale cress)
  ORGANISM  Arabidopsis thaliana
```

Parsing GenBank files

Load a Biopython GenBank parser and iterator

```
from Bio import GenBank
parser = GenBank.FeatureParser()
iterator = GenBank.Iterator(gb_handle, parser)
```

Loop through all records and keep track of Myb gene names

```
arabidopsis_names = []
while 1:
    record = iterator.next()
    if not(record):
        break
    if record.name[:2] == "At" and record.name not in arabidopsis_names:
        arabidopsis_names.append(record.name)
```

Load Microarray data

Data organized in a tab delimited file as expression ratios

Gene	start	10 min	30 min	1 hour	3 hour	6 hour
At1g01100	0.0	0.913	1.350	1.066	1.132	0.701
At1g01260	0.0	0.991	0.717	1.186	1.509	0.779
At1g01470	0.0	1.593	1.220	1.448	1.235	1.077
At1g01720	0.0	1.197	0.712	1.637	1.180	0.906

Input data from file and extract information of interest.

```
from Bio import Cluster

data_parts = Cluster.readdatafile("chitin_data.txt")
values = data_parts[0]
geneids = data_parts[2]
experiments = data_parts[6]
```

Selecting Myb transcription factors

Python code to look through the expression results and pick out the Myb expression results we are interested in

```
myb_values = []
myb_geneids = []
for cur_name in arabidopsis_names:
    try:
        cur_index = geneids.index(cur_name)
        myb_values.append(values[cur_index])
        myb_geneids.append(cur_name)
    except ValueError:
        pass
```

Performing microarray clustering

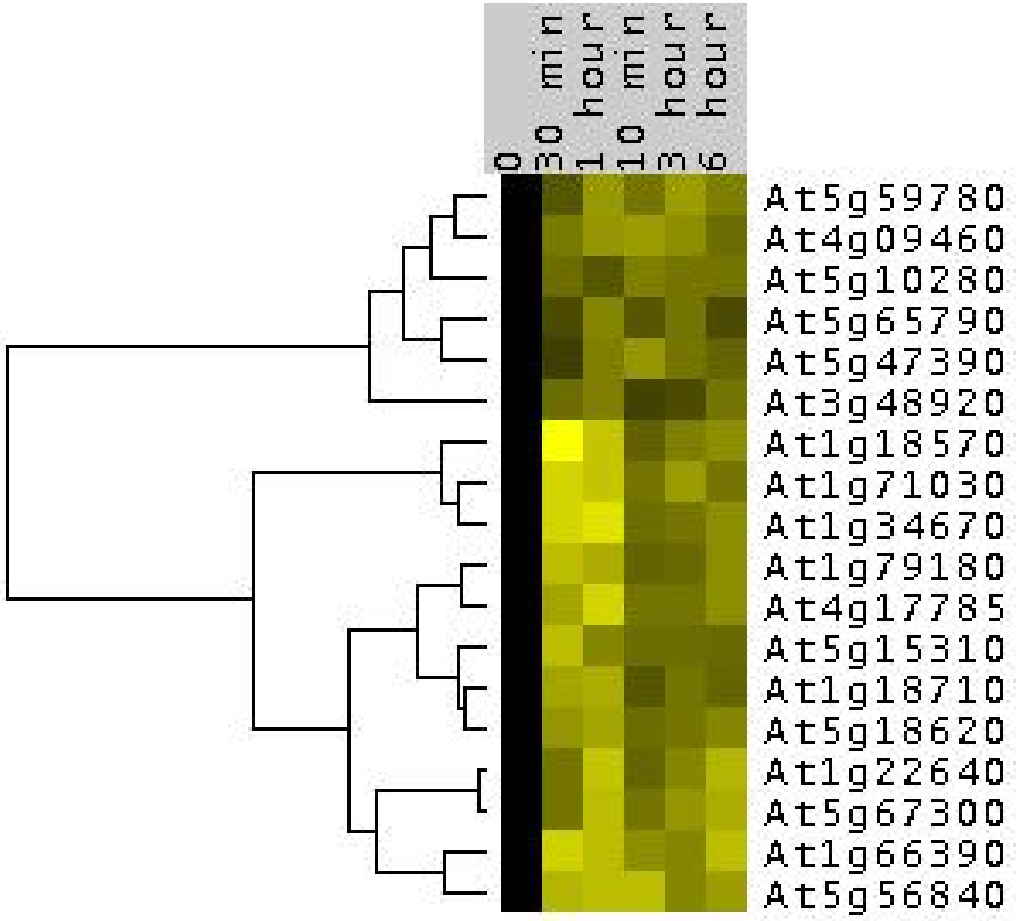
Hierarchical clustering of genes and experiments using Cluster package

```
gene_clusters, gene_linkdist = Cluster.treecluster(myb_values)
exp_clusters, exp_linkdist = Cluster.treecluster(myb_values, transpose = 1)
```

Output results for display by a viewing program

```
Cluster.writeclusterfiles("myb_chitin", myb_values,
                          myb_geneids, experiments,
                          geneclusters = gene_clusters,
                          genelinkdist = gene_linkdist,
                          expclusters = exp_clusters,
                          explinkdist = exp_linkdist)
```

Clustering results



Improving analysis pipelines

- Further development of component parts (more parsers, more web retrieval, more programs).

- Generalization allows more flexibility in designing and changing analyses
  - Open Biological Database Access (OBDA) – generalized retrieval.
  - Common sequence (`Bio.Seq`, `Bio.SeqRecord`) and analysis formats to parse into.
  - Mechanisms for parsing flat files into common formats (Martel plus standard XML tags).
  - Standard ways to call commandline programs.
  - Bio-Pipeline (`http://www.biopipe.org`) – Generalizing interactions between components.

Contributing to Biopython

- Lots of work to do.
- As with all open source projects, volunteers are always needed.
- Plenty of different things that need work
  - Coding support for new programs, file formats, databases
  - Adding tests for existing code
  - Updating and maintaining current code (bug fixes, speed-ups)
  - Documentation, Documentation, Documentation
  - News about what is going on with the project

Advantages of contributing to open-source projects

- Psychological – fuzzy warm feeling of donating your time to a useful project

- Practical (what is this going to do for me?)
  - Improve your coding skills (practice)
  - People look at your code, offer suggestions, bug fixes
  - Integrate with a community of people
  - Demonstrate good software skills
    - Being able to produce working code
    - Working in a community coding environment (CVS. . .)
    - Writing tests and documentation

Acknowledgments

All of the Biopython folks: Sebastian Bassi, Yair Benita, Peter Bienstman, Jeff Chang, Gavin Crooks, Andrew Dalke, Michiel de Hoon, Iddo Friedberg, Thomas Hamelryck, Michael Hoffman, Andreas Kuntzagk, Katharine Linder, Tarjei Mikkelsen, Thomas Sicheritz-Ponten and many more. . .